Performance Evaluation of Data Distributions with Load-Balancing for Sparse Arrays

Chun-Yuan Lin
Institute of Molecular and Cellular Biology, National Tsing Hua University, Taiwan, ROC
Email: cyulin@mx.nthu.edu.tw

Yeh-Ching Chung
Department of Computer Science, National Tsing Hua University, Taiwan, ROC
Email: ychung@cs.nthu.edu.tw

Jen-Shiuh Liu
Department of Information Engineering, Feng Chia University, Taiwan, ROC
Email: jsliu@fcu.edu.tw

Abstract

In our previous work, we have studied three data distribution schemes, Send Followed Compress (SFC), Compress Followed Send (CFS), and Encoding- Decoding (ED), for sparse arrays based on the traditional matrix representation (TMR) scheme. Since multi-dimensional arrays can also be represented by the extended Karnaugh map representation (EKMR) scheme, in this paper, we first apply the SFC/CFS/ED schemes based on the EKMR scheme. Then, we compare the performance of these three schemes with those based on the TMR scheme. Both theoretical analysis and experimental test were conducted. In theoretical analysis, we analyze the SFC/CFS/ED schemes based on the TMR/EKMR schemes in terms of the data distribution time and the data compression time. In experimental test, we implement these three schemes on an IBM SP2 parallel machine. The theoretical analysis and experimental results first show that the ED scheme is superior to the CFS scheme that is superior to the SFC scheme. Second, these three schemes based on the EKMR scheme outperform those based on the TMR scheme.

1. Introduction

Array operations are useful in a large number of important scientific codes, such as molecular dynamics [4], finite-element methods [9], climate modeling [19], atmosphere and ocean sciences [5], etc. Many data parallel programming languages, such as Fortran D [7], High Performance Fortran (HPF) [10], and Vienna Fortran [20], have been proposed and used by users to write efficient data parallel programs. However, it is a challenging problem to provide an efficient data distribution for irregular problems [18] on distributed memory multicomputers. In the literature [3, 20, 21], many methods have been proposed and were all performed in the following order, the data partition phase, then the data distribution phase, followed by the data compression phase, and called the *Send Followed Compress* (*SFC*). These methods are all focused on sparse arrays based on the *traditional matrix representation* (*TMR*) [12]. Since parallel multi-dimensional array operations [3, 5, 11, 12, 14-16] have been an extensively investigated problem, to propose efficient data distribution schemes for them becomes an important issue.

In our previous work [13], we have proposed the *Compress Followed Send* (*CFS*) and the *Encoding-Decoding* (*ED*) schemes based on the *TMR* scheme. The *extended Karnaugh map representation* (*EKMR*) [12] is another array representation scheme for multi-dimensional arrays. We also have proposed the *EKMR-Compressed Row Storage* (*ECRS*) and the *EKMR-Compressed Column Storage* (*ECCS*) [15] data compression methods based on the *EKMR* scheme. We have shown that some sparse array operations based on the *ECRS*/*ECCS* methods outperform those based the *CRSCCS* [2] methods based on the *TMR* scheme. Hence, in this paper, first, we apply the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme. Then, we compare the performance of them based on the *EKMR* scheme with those based on the *TMR* scheme.

In order to evaluate these three schemes, in the data partition phase, the 2D mesh partition [14] with load-balancing method is used. In this paper, we use the load-balancing method proposed by Zapata *et al*. [20]. In the data distribution phase, local sparse arrays are sent to processors in sequence. In the data compression phase, the *ECRS/ECCS* methods or formats are used for the *SFC*/*CFS*/*ED* schemes.

Both theoretical analysis and experimental test were conducted. In theoretical analysis, we analyze the *SFC*/*CFS*/*ED* schemes based on the *TMR*/*EKMR* schemes in terms of the data distribution/data compression time. Here, we do not consider the data partition time since the comparisons of these three schemes are all based on the

Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)
1087-4089/04 $ 20.00 © 2004 IEEE

same partition method. In experimental test, we implement these three schemes on an IBM SP2 parallel machine. The theoretical analysis and the experimental results show that the *ED* scheme outperforms the *CFS* scheme that outperforms the *SFC* scheme. Besides, the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme outperform those based on the *TMR* scheme.

2. Related Work

Many methods have been proposed to implement data distributions of sparse arrays in the literature. Zapata *et al.* [20] have proposed *Block Row Scatter* (*BRS*) and *Multiple Recursive Decomposition* (*MRD*) schemes for two-dimensional sparse arrays. In the *BRS*/*MRD* schemes, the data partition phase is performed first, then the data distribution phase, followed by the data compression phase. Ziantz *et al.* [21] proposed a run-time technique that was applied to sparse arrays for array distributions and off-processor data fetching to reduce the communication and computation time. They used the block data distribution scheme with a bin-packing algorithm to distribute a global sparse array to processors. Lee *et al*. [3] presented a library to speed up sparse array computations with Fortran 90 [1] array intrinsic functions for multi-dimensional arrays. They provided a data distribution scheme by extending the *MRD* scheme to multi-dimensional sparse arrays. Therefore, these schemes above all belong to the *SFC* scheme.

3. Preliminary Concepts

We describe the *EKMR* scheme and the *ECRS*/*ECCS* methods for three-dimensional arrays. The details of them for multi-dimensional arrays can be found in [12, 15]. We use the *TMR*(*n*)/*EKMR*(*n*) for an *n*-dimensional array based on the row-major data layout [14]. In the *EKMR* scheme, a multi-dimensional array is represented by a set of two-dimensional arrays. Let *A*[*k*][*i*][*j*] denote a 3×4×5 array based on the *TMR*(3). The corresponding array *A*′[4][15] based on the *EKMR*(3) is shown in Figure 1. The *ECRS*/*ECCS* methods use a set of three one-dimensional arrays *R*, *CK*, and *V* to compress a multi-dimensional sparse array based on the *EKMR* scheme. Given a sparse array *A* based on the *EKMR*(3), the *ECRS* (*ECCS*) method compresses all of non-zero array elements along the rows (columns for *ECCS*). Array *R* stores the number of non-zero array elements of each row (column for *ECCS*). The number of non-zero array elements in the *i*th row (*j*th column for *ECCS*) can be obtained by subtracting the value of *R*[*i*] from *R*[*i*+1]. The column (row for *ECCS*) indices and the values of non-zero array elements are stored in arrays *CK* and *V* store. The base of these three arrays is 0. An example of the *ECRS*/*ECCS* methods is given in Figure 2.

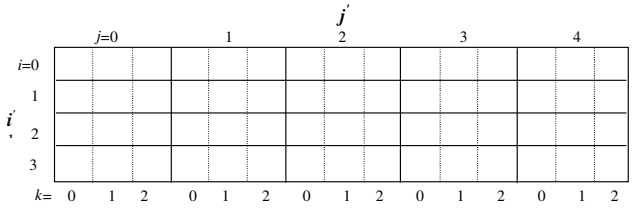


Figure 1: An array based on the *EKMR*(3).

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

Figure 3: A sparse array *A*′ based on the *EKMR*(3).

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

We describe the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme. Assume that a sparse array *A*′ based on the *EKMR*(3) shown in Figure 3 is stored in a host processor. Our goal is to distribute array *A*′ to 2×2 processor array.

4.1 The *SFC* Scheme

In the data partition phase, array *A*′ is partitioned into four local sparse arrays shown in Figure 4(a). In theory, each local sparse array has the same number of non-zero array elements. However, it may be impossible as Figure 4(a) in practice. The reason is that we can not partition a row or a column into two parts according to the load-balancing method. Since array elements of local sparse arrays are not stored in consecutive memory locations, they need to be packed before sending to processors. In the data distribution phase, packed local sparse arrays are sent to processors in sequence. Figure 4(b) shows the local sparse array received by each processor. In the data compression phase, the received local sparse array is compressed by the *ECCS* method in Figure 4(c).

4.2 The *CFS* Scheme

The three phases are performed in the following order, the data partition phase, then data compression phase, followed by the data distribution phase. In the data partition and data compression phases, the processes are the same as those of the *SFC* scheme. However, the values stored in array *CK* are global array indices. In the data distribution phase, arrays *R*, *CK*, and *V* of each local sparse array are packed and then sent to its corresponding processor. Each processor unpacks the buffer to get arrays *R*, *CK*, and *V*. Each processor $P_{i,j}$ converts the values stored in array *CK* to local array indices by subtracting α from them, where α is the total number of columns (rows for *ECCS*) in processors $P_{i,0}$, $P_{i,1}$, …, $P_{i,j-1}$

($P_{0,j}$, $P_{1,j}$, …, $P_{i-1,j}$ for *ECCS*). An example of the *CFS* scheme for array A' using the *ECCS* method is given in Figure 5. Figure 5(c) only shows the data distribution phase for processor $P_{1,0}$.

4.3 The *ED* Scheme

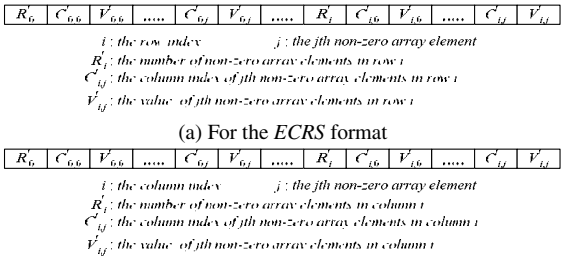
The data compression phase can be divided into two steps, encoding and decoding. The data partition phase is performed first, then the encoding step, followed by the data distribution phase and the decoding step. In the data partition phase, the process is the same as that of the *SFC* scheme. In the encoding step, each local sparse array is encoded into a buffer *B* shown in Figure 6. Each $C'_{i,j}$ is a global array index. In the data distribution phase, these buffers *B* are sent to processors in sequence. In the decoding step, a buffer *B* is decoded to get arrays *R*, *CK*, and *V* in each processor. To get array *R*, *R*[0] is first initialized to 1. Then, other values of array *R* are computed according to the formula $R[i+1]=R[i]+R'_i$. To get arrays *CK* and the *V*, we move all $C'_{i,j}$ and $V'_{i,j}$ stored in buffer *B* to arrays *CK* and *V*. Each processor $P_{i,j}$ converts each $C'_{i,j}$ to the local array index by subtracting β from them, where β is the total number of columns (rows for *ECCS*) in processors $P_{i,0}$, $P_{i,1}$, …, $P_{i,j-1}$ ($P_{0,j}$, $P_{1,j}$, …, $P_{i-1,j}$ for *ECCS*). An example of the *ED* scheme for array A' using the *ECCS* format is given in Figure 7. Figure 7(d) only shows the decoding step for processor $P_{1,0}$. For the four- or higher dimensional sparse arrays, the *SFC*/*CFS*/*ED* schemes are similar to those for the three-dimensional sparse array.

5. Theoretical Analysis

Due to page limitation, we only analyze the performance for the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme with the *ECRS* method. In Table 1, we list the notations used in the theoretical analysis. Assume that an n^3 sparse array A' based on the *EKMR* scheme is stored in a host processor and we want to distribute array A' to $r\times q$ processors. The number of non-zero array elements in array A' is sn^3 and we assume that the sparse probability [8] for each array element is equal. The number of non-zero array elements in each local sparse array is $sn^3/r\times q$. The largest local sparse array is $\alpha' n^3$.

5.1 The *SFC* Scheme

In the data distribution phase, each local sparse array is packed and then sent to a processor sequentially. $T_{Distribution}=r\times q\times T_{Startup}+n^3\times T_{Data}+ n^3\times T_{Operation}$. In the data compression phase, local sparse arrays in all processors are compressed simultaneously. $T_{Compression}= n^3\times(\alpha'+3/r\times qs)\times T_{Operation}$.



(a) For the *ECRS* format

(b) For the *ECCS* format

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

Table 1: The notations are used in this paper.

Notation	*Descriptions*
$T_{Startup}$	*The startup time for a communication channel*
T_{Data}	*The transmission time for sending an array element*
$T_{Operation}$	*The average time of an array element to do an operation.*
$T_{Distribution}$	*The data distribution time for the data distribution phase.*
$T_{Compression}$	*The data compression time for the data compression phase.*
A	*A multi-dimensional sparse array based on the EKMR scheme*
$r\times q$ (p)	*The number of processors*
s	*The sparse ratio of a global sparse array*
$\alpha=\{\alpha_i \mid i=0,1,\ldots,p-1\}$	*A set of space ratios of local sparse arrays. The space ratio of the largest local sparse array is denoted as* α' *and the size is* $r'\times q'$.

Table 2: The data distribution/data compression time.

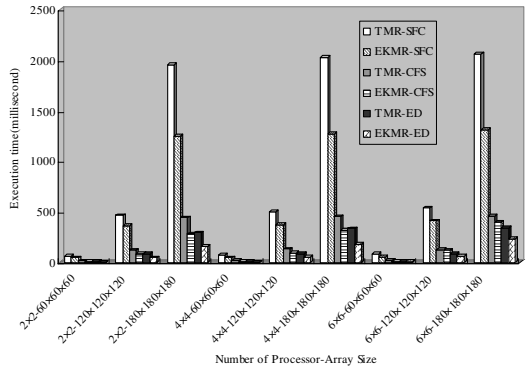
Scheme	*Method*	*Complexity*	*Cost*
TMR	*SFC*	$T_{Distribution}$	$r\times q\times T_{Startup} + N\times T_{Data}+ N\times T_{Operation}$
		$T_{Compression}$	$(N\times(\alpha+Ms))\times T_{Operation}$
	CFS	$T_{Distribution}$	$r\times q\times T_{Startup} + (ksN+qn+rq)\times T_{Data} +$ $(N\times(k+M)s+r'+qn+rq+1)\times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

5.2 The *CFS* Scheme

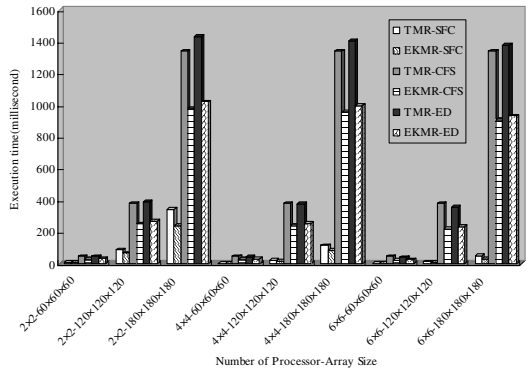
In the data compression phase, each local sparse array is compressed sequentially. $T_{Compression}=n^3\times(1+3s)\times T_{Operation}$. In the data distribution phase, each local compressed array is packed and then sent to a processor sequentially. These buffers are unpacked to get arrays *R*, *CK*, and *V* simultaneously. $T_{Distribution} = r\times q\times T_{Startup} + (2n^3s+qn+rq)\times T_{Data}+ (n^3\times(2+3/r\times q)s+r'+qn+rq+1)\times T_{Operation}$.

5.3 The *ED* Scheme

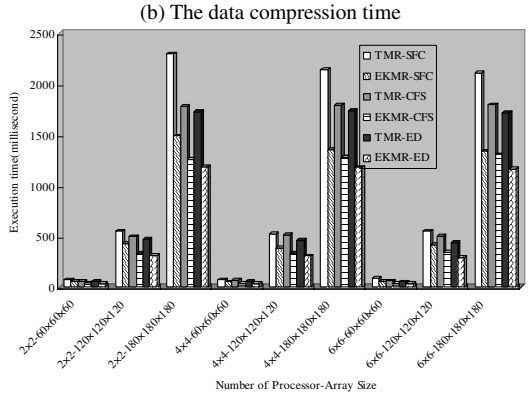
In the encoding step, each local sparse array is encoded into the buffer *B* sequentially. In the data distribution phase, each buffer *B* is sent to a processor sequentially. $T_{Distribution}=r\times q\times T_{Startup}+(2n^3s+qn)\times T_{Data}$. In the decoding step, these buffers are decoded to get arrays *R*, *CK*, and *V* simultaneously. $T_{Compression} = (n^3\times(1+(3+3/r\times q)s)+r'+1)\times T_{Operation}$. Assume that an n^k sparse array A', where $k > 2$, is stored in a host processor. Table 2 lists the data distribution/data compression time. In order to simplify the results, we use symbols *N*, *M*, and *Q* to substitute symbols n^k, $(k+1)/r\times q$, and $3/r\times q$, respectively.



(a) The data distribution time



(b) The data compression time



(c) The sum of the data distribution/data compressing time

Figure 8: The data distribution/data compression time.

5.4 Discussions

From Table 2, for the *EKMR* scheme, first, we can see that $T_{Distribution}(ED)<T_{Distribution}(CFS)$. Second, if s is less than 0.5, we can see that $T_{Distribution}(ED)$ and $T_{Distribution}(CFS)$ is less than $T_{Distribution}(SFC)$. In [15], we have shown that s must be less than 0.5 if we want to use the *ECRS*/*ECCS* methods. Moreover, it shows that over 80% sparse array applications in which s is less than 0.1 according to the Harewell-Boeing Sparse Matrix Collection [5]. Hence, we have four **Remark**.

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

2. $T_{Compression}(SFC) < T_{Compression}(CFS) < T_{Compression}(ED)$.

3. The *ED* scheme outperforms the *CFS* scheme.

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

In general, T_{Data} is larger than or equal to $T_{Operation}$ on a distributed memory multicomputer. If we assume that T_{Data} is equal to $T_{Operation}$, the conditions are rewritten to $s < (1+\alpha')/5$ and $s<(1+\alpha')/7$. Since s in practical applications is very small (< 0.1), the conditions can be satisfied easily.

From Table 2, for the *TMR*/*EKMR* schemes, we have three **Remark**.

5. The data distribution time of the *SFC*/*CFS*/*ED* scheme based on the *EKMR* scheme is less than that based on the *TMR* scheme.

6. The data compression time of the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme is less than that based on the *TMR* scheme.

7. The *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme outperform those based on the *TMR* scheme.

The reasons are two-fold. First, for the *SFC* scheme, the *EKMR* scheme can reduce the costs of packing non-continuous data blocks [14]. Second, for these three schemes, the time required to compress a sparse array can be reduced since the number of one-dimensional arrays used by the *ECRS*/*ECCS* methods does not increase as the dimension increases [15].

6. Experimental Results

In experimental test, we implement the *SFC*/*CFS*/*ED* schemes based on the *TMR*/*EKMR* schemes on an IBM SP2 parallel machine. All programs are written in *C* + MPI (*Message Passing Interface*) [21] codes. The sparse ratio is set to 0.1 for all test three-dimensional sparse arrays used as test samples.

Figure 8 shows the data distribution/data compression time of the *SFC*/*CFS*/*ED* schemes based on the *TMR* scheme with the *CRS* method and the *EKMR* scheme with the *ECRS* method using the 2D mesh partition with load-balancing method. For the *EKMR* scheme, from Figure 8(a), the result matches Remark 1. The reasons are two-fold. First, for the *CFS*/*ED* schemes, we do not send entire local sparse arrays to processors. Second, local compressed arrays do not need to be packed for the *ED* scheme. From Figure 8(b), the result matches Remark 2. The reason is that we do not compress entire global sparse array for the *SFC* scheme. From Figure 8(c), the result matches Remarks 3 and 4. The reason is that the conditions, $T_{Data} > (3\text{-}10\alpha'/8)T_{Operation}$ and $T_{Data} > (5\text{-}10\alpha'/8)T_{Operation}$, are satisfied. From Figure 8 and Table 2, we can estimate that T_{Data} is close to $1.2\times T_{Operation}$. For the *TMR*/*EKMR* schemes, from Figure 8, these results match Remarks 5, 6, and 7. From Figure 8, we can see that the experimental results match the theoretical analysis shown in Table 2.

7. Conclusions

In this paper, first, we applied the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme. Then, we have compared the performance of these three schemes with those based on the *TMR* scheme. Both theoretical analysis and experimental test were conducted. From the theoretical analysis and the experimental results, we can see that the *ED* scheme outperforms the *CFS* scheme that outperforms the *SFC* scheme. Moreover, the *SFC*/*CFS*/*ED* schemes based on the *EKMR* scheme outperform those based on the *TMR* scheme.

Acknowledgements

The work in this paper was partially supported by National Science Council of the Republic of China under contract NSC91-2213-E-007-104.

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. 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*, SIAM, 1994.

[3] R.G. Chang, T.R. Chung, and J.K. Lee, "Parallel Sparse Supports for Array Intrinsic Functions of Fortran 90," *J. Supercomputing*, vol. 18, no. 3, Mar. 2001, pp. 305-339.

[4] J.K. Cullum and R.A. Willoughby, *Lanczos Algorithms for Large Symmetric Eignenvalue Computations*, Birkhauser Boston, 1985.

[5] C.H.Q. Ding, "An Optimal Index Reshuffle Algorithm for Multidimensional Arrays and Its Applications for Parallel Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 12, no. 3, Mar. 2001, pp. 306-315.

[6] 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.*

[7] 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 Univ.*

[8] B.B. Fraguela, R. Doallo, and E.L. Zapata, "Cache Probabilistic Modeling for Basic Sparse Algebra Kernels Involving Matrices with a Non-Uniform Distribution," *Proc. IEEE Euromicro Conf.*, 1998, pp. 345-348.

[9] G.H. Golub and C.F. Van Loan, *Matrix Computations,* 2nd Edition, The John Hopkins University Press, Baltimore, Maryland 21218, 1989.

[10] High Performance Fortran Forum, *High Performance Fortran Language Specification*, Rice Univ.**,** 1997.

[11] M. Kandemir, J. Ramanujam, and A. Choudhary, "Improving Cache Locality by a Combination of Loop and Data Transformations," *IEEE Trans. Computers*, vol. 48, no. 2, Feb. 1999, pp. 159-167.

[12] C.Y. Lin, J.S. Liu, and Y.C. Chung, "Efficient Representation Scheme for Multi-Dimensional Array Operations," *IEEE Trans. Computers,* vol. 51, no. 3, Mar. 2002, pp. 327-345.

[13] C.Y. Lin, Y.C. Chung, and J.S. Liu, "Data Distribution Schemes of Sparse Arrays on Distributed Memory Multicomputers," *Proc. ICPP Workshops Compile/Runtime Techniques for Parallel Computing*, 2002, pp. 551~558.

[14] 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. Parallel and Distributed Systems,* vol. 14, no. 7, July 2003, pp. 625-639.

[15] C.Y. Lin, Y.C. Chung, and J.S. Liu, "Efficient Data Compression Methods for Multi-Dimensional Sparse Array Operations," *IEEE Trans. Computers*, vol. 52, no. 12, December 2003, pp. 1640-1646.

[16] J.S. Liu, C.H. Huang, and D.Y. Yang, "Parallel Volume Rendering with Sparse Data Structures," *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and System*, Nov. 2002, pp. 594-599.

[17] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, University of Tennessee, June 1995.

[18] R. Ponnusamy, J. Saltz, R. Das, C. Koelbel, A. Choudhary. "Embedding Data Mappers with Distributed Memory Machine Compilers," *ACM SIGPLAN Notices*, vol. 28, no. 1, 1993, pp. 52-55.

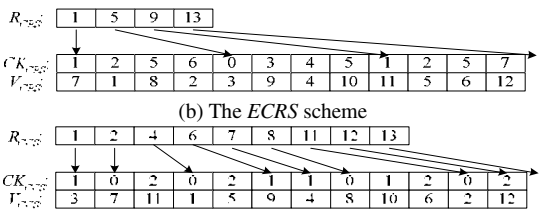
[19] M. Ujaldon, E.L. Zapata, S.D. Sharma, and J. Saltz, "Parallelization Techniques for Sparse Matrix Applications," *J. parallel and distribution computing*, vol. 38, no. 2, Nov. 1996, pp. 256-266.

[20] 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. Parallel and Distributed Systems*, vol. 8, no. 10, Oct. 1997, pp. 1068-1083.

[21] L.H. Ziantz, C.C. Ozturan, and B.K. Szymanski, "Run-Time Optimization of Sparse Matrix-Vector Multiplication on SIMD Machines," *Proc. Int'l Conf. Parallel Architectures and Languages*, 1994, pp. 313-322.

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

(a) A sparse array based on the *EKMR*(3)



(b) The *ECRS* scheme

(c) The *ECCS* scheme

Figure 2: The *ECRS*/*ECCS* methods based on the *EKMR*(3).

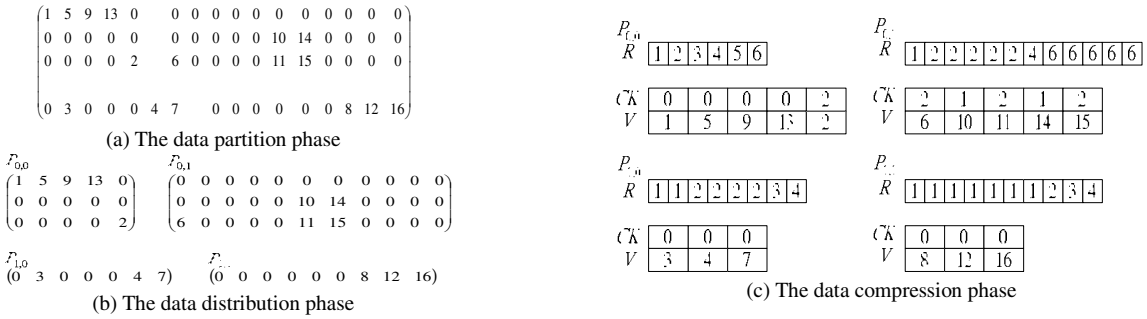


Figure 4: An example of the *SFC* scheme for array A'.

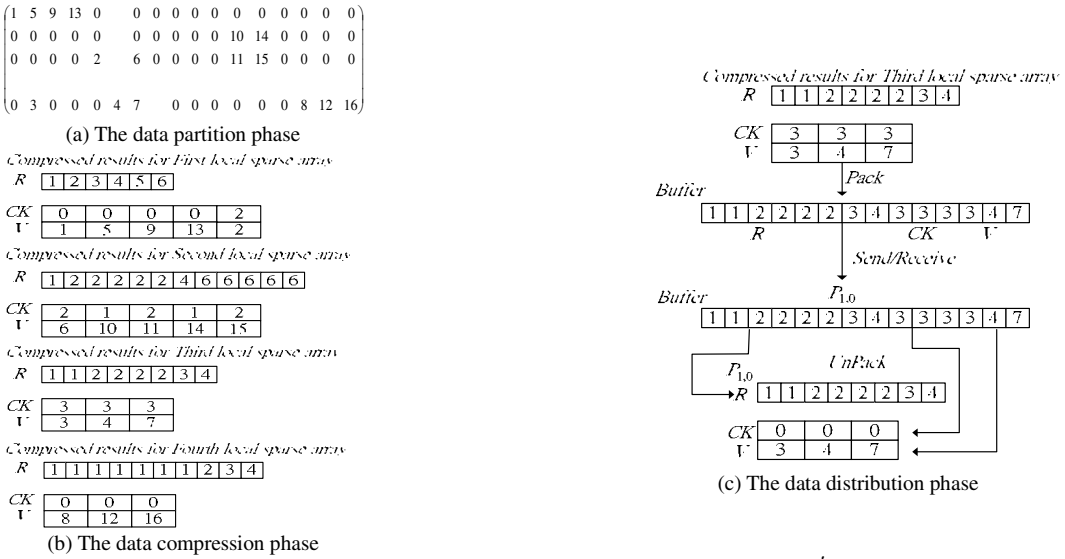


Figure 5: An example of the *CFS* scheme for array A'.

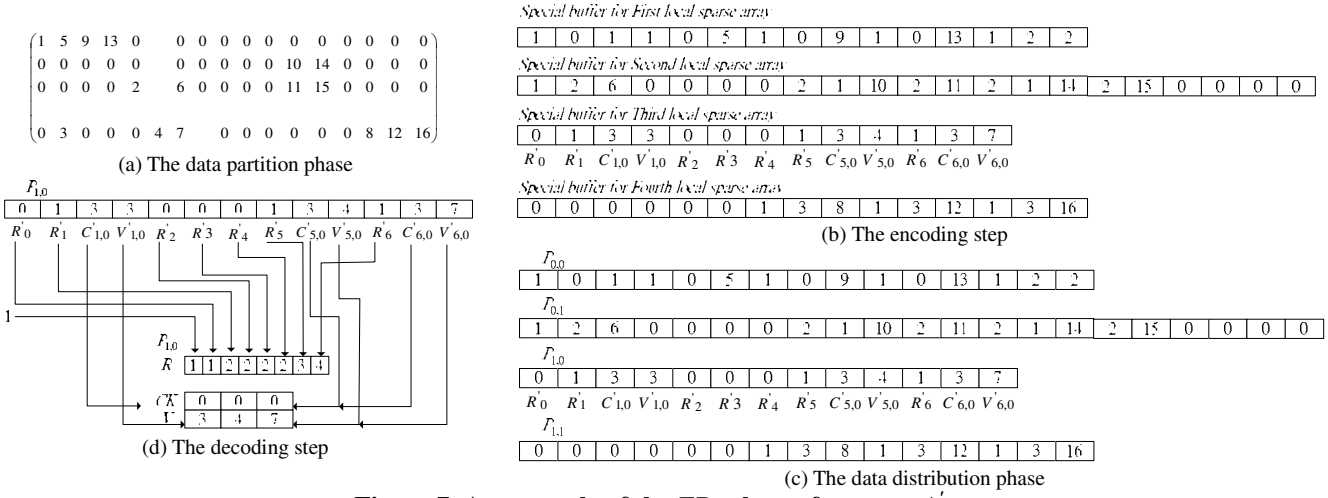


Figure 7: An example of the *ED* scheme for array A'.