Efficient Data Compression Methods for Multidimensional Sparse Array Operations Based on the *EKMR* Scheme

Chun-Yuan Lin, *Member*, *IEEE Computer Society*,
Yeh-Ching Chung, *Member*,
IEEE Computer Society, and
Jen-Shiuh Liu

Abstract—In our previous work, we have proposed the *extended Karnaugh map representation* (*EKMR*) scheme for multidimensional array representation. In this paper, we propose two data compression schemes, *EKMR Compressed Row/Column Storage* (*ECRS*/*ECCS*), for multidimensional sparse arrays based on the *EKMR* scheme. To evaluate the proposed schemes, we compare them to the *CRS*/*CCS* schemes. Both theoretical analysis and experimental tests were conducted. In the theoretical analysis, we analyze the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes in terms of the time complexity, the space complexity, and the range of their usability for practical applications. In experimental tests, we compare the compressing time of sparse arrays and the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* based on the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes. The theoretical analysis and experimental results show that the *ECRS*/*ECCS* schemes are superior to the *CRS*/*CCS* schemes for all the evaluated criteria, except the space complexity in some cases.

Index Terms—Data compression scheme, sparse array operation, multidimensional sparse array, Karnaugh map, sparse ratio.

1 Introduction

Array operations are useful in a large number of important scientific codes, such as molecular dynamics [4], finite-element methods [7], climate modeling [12], etc. For sparse array operations, in general, sparse arrays are compressed by some data compression schemes in order to obtain better performance. The *Compressed Row Storage* (*CRS*) [2] and the *Compressed Column Storage* (*CCS*) [2] (or *Compressed Sparse Column/Row* [14]) are common used schemes [3], [5], [8], [11], [13], [15], [16] due to their simplicity and purity with a weak dependence relationship between array elements in a sparse array.

A multidimensional array can be viewed as a collection of two-dimensional arrays. This scheme is called *traditional matrix representation* (*TMR*) [10]. The *CRS*/*CCS* schemes are both based on the *TMR* scheme. For the *CRS*/*CCS* schemes, a two-dimensional sparse array can be compressed into three one-dimensional arrays. However, for higher dimensional sparse arrays, sparse array operations based on *CRS*/*CCS* schemes usually do not perform well. The reasons are two-fold. First, the number of one-dimensional arrays used to compress sparse arrays increases as the dimension increases because more one-dimensional arrays are needed to store extra indices of nonzero array elements. This increases the time and the memory space of compressing a sparse array. Second, the costs of indirect data access [1] and index comparisons for sparse array operations increase as the dimension increases.

In our previous work [10], we have proposed the *extended Karnaugh map representation* (*EKMR*) scheme for multidimensional array representation and have shown that dense array operations based on the *EKMR* scheme have better performance than those based on the *TMR* scheme. In this paper, we propose the *EKMR Compressed Row/Column Storage* (*ECRS*/*ECCS*) data compression schemes for multidimensional sparse arrays based on the *EKMR* scheme. Given a k-dimensional sparse array with a size of m along each dimension, the $EKMR(k)$ can be represented by $m^{k-4}EKMR(4)$. For k = 3 or 4, the *ECRS*/*ECCS* schemes use three one-dimensional arrays to compress the sparse array. For $k > 4$, the *ECRS*/*ECCS* schemes first use three one-dimensional arrays to compress $m^{k-4}EKMR(4)$ sparse arrays individually. Then, an abstract pointer array with a size of m^{k-4} is used to link these three arrays in each $EKMR(4)$.

To evaluate the proposed schemes, we compare them to the *CRS*/*CCS* schemes. Both theoretical analysis and experimental tests were conducted. In the theoretical analysis, we analyze the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes in terms of the time complexity, the space complexity, and the range of their usability for practical applications. The theoretical analysis shows that the time complexities of the *ECRS*/*ECCS* schemes are less than those of the *CRS*/*CCS* schemes. For most of sparse arrays in practical applications, the space complexities of the *ECRS*/*ECCS* schemes are less than those of the *CRS*/*CCS* schemes. The range of usability of the *ECRS*/*ECCS* schemes is wider than that of the *CRS*/*CCS* schemes for practical applications. In experimental tests, we compare the compressing time of sparse arrays and the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* based on the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes. The experimental results show that the compressing time of the *ECRS*/*ECCS* schemes is less than that of the *CRS*/*CCS* schemes and sparse array operations based on the *ECRS*/*ECCS* schemes outperform those based on the *CRS*/*CCS* schemes. The reasons are two-fold. First, for the *CRS*/*CCS* schemes, the number of one-dimensional arrays used to compress sparse arrays increases as the dimension increases, while the *ECRS*/*ECCS* schemes do not. The compressing time required by the *ECRS*/*ECCS* schemes is less than that by the *CRS*/*CCS* schemes. Second, the costs of the indirect data access and index comparisons for sparse array operations based on the *ECRS*/*ECCS* schemes are less than those based on the *CRS*/*CCS* schemes.

This paper is organized as follows: In Section 2, we will briefly describe the*EKMR* scheme. Section 3 will describe the *ECRS*/*ECCS* schemes in detail. The theoretical analysis of the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes will be given in this section as well. The experimental results will be given in Section 4.

2 The *EKMR* Scheme

In this section, we briefly describe the *EKMR* scheme before presenting the *ECRS*/*ECCS* schemes. The details of the *EKMR* scheme can be found in [10]. In the following, we use *TMR*(n) and *EKMR*(n) for the *TMR* and the *EKMR* schemes of an n-dimensional array based on the row-major data layout, respectively.

The idea of the *EKMR* scheme is based on the Karnaugh map. When n = 1 and 2, the *TMR* and the *EKMR* schemes are the same. Let $A[k][i][j]$ denote a $3 \times 4 \times 5$ array based on the *TMR*(3). The corresponding *EKMR*(3) of array A is shown in Fig. 1. The *EKMR*(3) is represented by a 4×15 two-dimensional array. In the *EKMR*(3), we use the index variable i' to indicate the row direction and the index variable j' to indicate the column direction. Note that the index i' is the same as the index i, whereas the index j' is a combination of indices j and k. The way to obtain the *EKMR*(4) is similar to that of the *EKMR*(3). The *EKMR*(4) is also represented by a two-dimensional array. In the *EKMR*(4), the index i' is a

- C.-Y. Lin and J.-S. Liu are with the Department of Information Engineering, Feng Chia University, Taichung, Taiwan 407, ROC. E-mail: {cylin, liuj}@iecs.fcu.edu.tw.
- Y.-C. Chung is with the Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan 300, ROC. E-mail: ychung@cs.nthu.edu.tw.

Manuscript received 13 Feb. 2001; revised 8 Oct. 2002; accepted 1 July 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 113613.

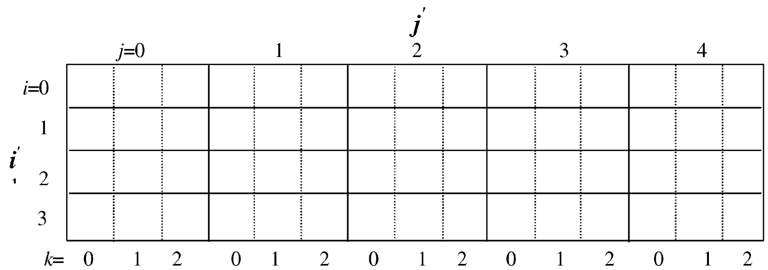

Fig. 1. The *EKMR*(3) scheme.

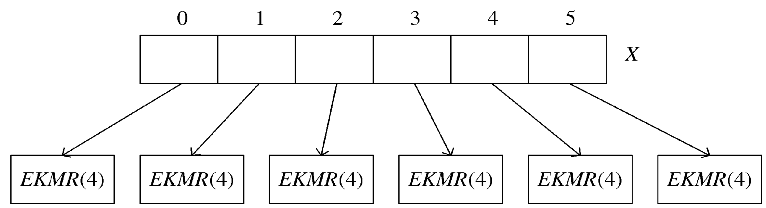

Fig. 2. An example of the *EKMR*(6).

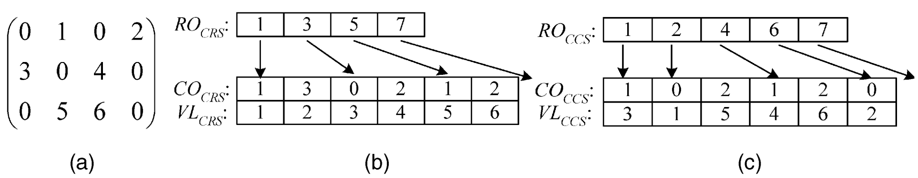

Fig. 3. The *CRS/CCS* schemes for a two-dimensional sparse array based on the *TMR*(2). (a) A sparse array. (b) The *CRS* scheme. (c) The *CCS* scheme.

combination of indices l and i, whereas the index j' is a combination of indices j and k. Based on the *EKMR*(4), we can generalize our results to n-dimensional arrays. The *EKMR*(n) can be represented by $m^{n-4}EKMR(4)$ and a one-dimensional array X with a size of m^{n-4} are used to link these *EKMR*(4). Fig. 2 shows a $3 \times 2 \times 2 \times 3 \times 4 \times 5$ six-dimensional array represented by six *EKMR*(4) with a size of 8×15.

3 THE *ECRS/ECCS* SCHEMES

We first describe the *CRS/CCS* schemes for sparse arrays based on the *TMR* scheme. Then, we present the *ECRS/ECCS* schemes for sparse arrays based on the *EKMR* scheme.

3.1 The *CRS/CCS* Schemes

Given a two-dimensional sparse array, the *CRS* (*CCS*) scheme using one one-dimensional floating-point array *VL* and two one-dimensional integer arrays *RO* and *CO* to compress all of the nonzero array elements along the rows (columns for *CCS*) of the sparse array. Array *RO* stores information about the nonzero array elements of each row (column for *CCS*). The number of nonzero array elements in the *i*th row (*j*th 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. The base of these three arrays is 0. An example of the *CRS/CCS* schemes for a two-dimensional sparse array is given in Fig. 3. Fig. 3a shows a 3×4 two-dimensional sparse array. Fig. 3b and Fig. 3c show the *CRS/CCS* schemes, respectively. In Fig. 3b, the number of nonzero array elements in the second row can be obtained by $RO_{CRS}[3] - RO_{CRS}[2] = 7 - 5 = 2$. The column indices of nonzero array elements of the second row are stored in $CO_{CRS}[RO_{CRS}[2]-1], \ldots, CO_{CRS}[RO_{CRS}[3]-2]$. The values of nonzero array elements of the second row are stored in $VL_{CRS}[4:5]$. Based on the *CRS/CCS* schemes above, a sparse array based on the *TMR*(3) can be compressed by adding one one-dimensional integer array *KO*. In the *CRS* (*CCS*) scheme, array *KO* stores the third dimension indices of nonzero array elements of each row (column for *CCS*). An example of the *CRS/CCS* schemes for a sparse array based on the *TMR*(3) is shown in Fig. 4. For four or higher dimensional sparse arrays based on the *TMR* scheme, more one-dimensional integer arrays are needed.

3.2 The *ECRS/ECCS* Schemes

The *ECRS/ECCS* schemes use one one-dimensional floating-point array *V* and two one-dimensional integer arrays *R* and *CK* to compress a multidimensional sparse array based on the *EKMR* scheme. Given a sparse array based on the *EKMR*(3), the *ECRS* (*ECCS*) scheme compresses all of nonzero array elements along the rows (columns for *ECCS*) of the sparse array. Array *R* stores information of nonzero array elements of each row (column for *ECCS*). The number of nonzero array elements in the *i*th row (*j*th column for *ECCS*) can be obtained by subtracting the value of *R*[i] from *R*[$i+1$]. Array *CK* stores the column (row for *ECCS*) indices of nonzero array elements of each row (column for *ECCS*). Array *V* stores the values of nonzero array elements. The base of these three arrays is 0. An example of the *ECRS/ECCS* schemes for a sparse array based on the *EKMR*(3) is given in Fig. 5. Fig. 5a shows a 3×8 sparse array based on the *EKMR*(3) whose *TMR*(3) is shown in Fig. 4a. Fig. 5b and Fig. 5c show the *ECRS/ECCS* schemes, respectively.

Similarly, we can use arrays *R*, *CK*, and *V* to compress a sparse array based on the *EKMR*(4) in the *ECRS/ECCS* schemes. Since *EKMR*(k) can be represented by $m^{k-4}EKMR(4)$, in the *ECRS/ECCS* schemes, each *EKMR*(4) is first compressed by using arrays *R*, *CK*, and *V*. Then, an abstract pointer array with a size of m^{k-4} is used to link arrays *R*, *CK*, and *V* in each *EKMR*(4). For example, assume that there is a $3 \times 2 \times 2 \times \times 4 \times 5$ sparse array *A* based on the *TMR*(6). The sparse array *A'* based on the *EKMR*(6) can be represented by six *EKMR*(4) with a size of 8×15. In the *ECRS/ECCS* schemes, we first compress each *EKMR*(4) to arrays *R*, *CK*, and *V*. Then, we use an abstract pointer array with a size of 6 to

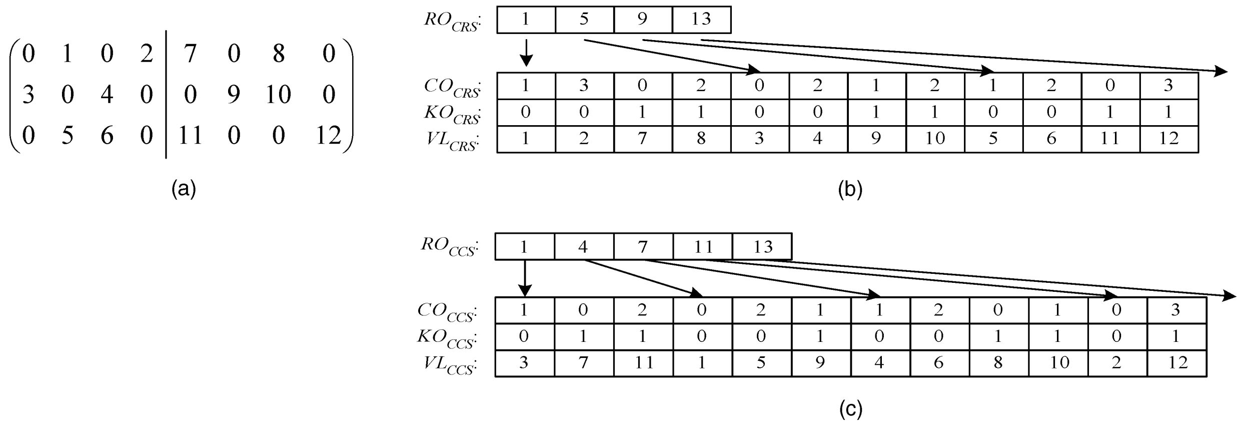

Fig. 4. The *CRS*/*CCS* schemes for a three-dimensional sparse array based on the *TMR*(3). (a) A sparse array based on the *TMR*(3). (b) The *CRS* scheme. (c) The *CCS* scheme.

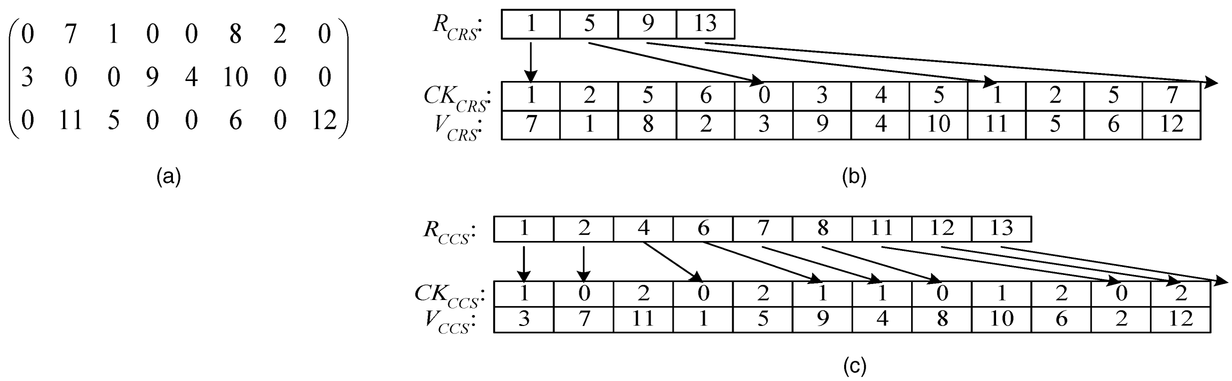

Fig. 5. The *ECRS*/*ECCS* schemes for a three-dimensional sparse array based on the *EKMR*(3). (a) A sparse array based on the *EKMR*(3). (b) The *ECRS* scheme. (c) The *ECCS* scheme.

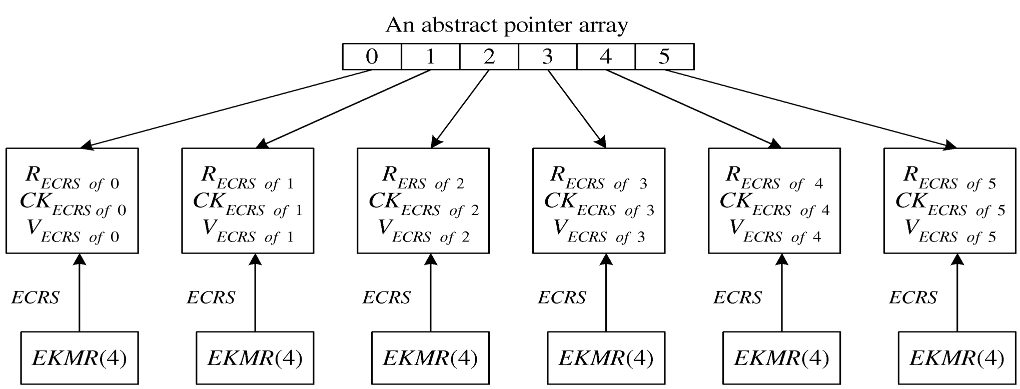

Fig. 6. The *ECRS* scheme for a six-dimensional sparse array based on the *EKMR*(6).

link arrays *R*, *CK*, and *V* of each *EKMR*(4). An example is shown in Fig. 6.

3.3 Theoretical Analysis

Assume that an n^3 sparse array *A* based on the *TMR*(3) with sparse ratio *S* is given. The number of nonzero array elements of sparse array *A* is Sn^3. We assume that the sparse probability [6] for each array element is equal. In the *CRS*/*CCS* schemes, four arrays, *RO*, *CO*, *KO*, and *VL*, are used to compress sparse array *A*. For the *CRS* (*CCS*) scheme, it first needs to scan the entire array *A* along the rows (columns for *CCS*) to find all of nonzero array elements. The cost of scanning entire array *A* is n^3. Then, it needs to record the information of each nonzero array element to these four arrays. The cost of recording the information to those four arrays is $4Sn^3$.Therefore, the time complexities of the *CRS*/*CCS* schemes both are $n^3 + 4Sn^3$.

Let sparse array A' be the corresponding sparse array *A* based on the *EKMR*(3). In the *ECRS*/*ECCS* schemes, three arrays, *R*, *CK*, and *V*, are used to compress sparse array A'. Therefore, the time complexities of the *ECRS*/*ECCS* schemes both are $n^3 + 3Sn^3$. For four or higher dimensional sparse arrays, we can obtain the time complexities of the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes in a similar manner. Table 1 lists the time complexities of the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes. In Table 1, the improved rate of compressing time (*IRC*) is defined as

$$IRC(\%) = ((T_{CRS/CCS} - T_{ECRS/ECCS})/T_{CRS/CCS}) \times 100,$$

TABLE 1
Time Complexities for the *CRS/CCS* and the *ECRS/ECCS* Schemes

Schemes / Dimensions	*CRS/CCS*	*ECRS/ECCS*	*IRC* (%)
3-*D*	n^3+4Sn^3	n^3+3Sn^3	$S/(1+4S)\times100$
4-*D*	n^4+5Sn^4	n^4+3Sn^4	$2S/(1+5S)\times100$
k-*D* ($k \geq 5$)	$n^k+(k+1)Sn^k$	n^k+3Sn^k	$(k-2)S/(1+(k+1)S)\times100$

TABLE 2
Space Complexities for the *CRS/CCS* and the *ECRS/ECCS* Schemes

Schemes / Dimensions	*CRS/CCS*	*ECRS/ECCS*	*IRS* (%)
3-*D*	$(2Sn^3+n+1)\alpha+Sn^3\beta$	*ECRS*: $(Sn^3+n+1)\alpha+Sn^3\beta$ *ECCS*: $(Sn^3+n^2+1)\alpha+Sn^3\beta$	*ECRS*: If $S>0$, *IRS* = $Sn^3\alpha/(2Sn^3+n+1)\alpha+Sn^3$ *ECCS*: If $S>1/n$, *IRS* = $(Sn^3-n^2+n)\alpha/(2Sn^3+n+1)\alpha+Sn^3$
4-*D*	$(3Sn^4+n+1)\alpha+Sn^4\beta$	$(Sn^4+n^2+1)\alpha+Sn^4\beta$	If $S>1/2n^2$, *IRS* = $(2Sn^4-n^2+n)\alpha/(3Sn^4+n+1)\alpha+Sn^4$
k-*D* ($k \geq 5$)	$((k-1)Sn^k+n+1)\alpha+Sn^k\beta$	$(Sn^k+n^{k-2}+n^{k-4})\alpha+Sn^k\beta$	If $S>1/(k-2)n^2$, *IRS* = $((k-2)Sn^k-n^{k-2}+n)\alpha/((k-1)Sn^k+n+1)\alpha+Sn^k$

TABLE 3
The Range of Usability of the *CRS/CCS* and the *ECRS/ECCS* Schemes

Schemes / Dimensions	*CRS/CCS*	*ECRS/ECCS*
3-*D*	$S<\beta/(2\alpha+\beta)$	$S<\beta/(\alpha+\beta)$
4-*D*	$S<\beta/(3\alpha+\beta)$	$S<\beta/(\alpha+\beta)$
k-*D* ($k \geq 5$)	$S<\beta/((k-1)\alpha+\beta)$	$S<\beta/(\alpha+\beta)$

where $T_{CRS/CCS}$ and $T_{ECRS/ECCS}$ are the compressing time of sparse arrays based on the CRS/CCS and the $ECRS/ECCS$ schemes, respectively.

In the CRS/CCS schemes, for sparse array *A*, the size of array *RO* is $n+1$, the size of arrays *CO*, *KO*, and *VL* is all Sn^3. Assume that an integer is α bytes long and a floating-point is β bytes long. The space complexities of the CRS/CCS schemes both are $(2Sn^3+n+1)\alpha+Sn^3\beta$. In the $ECRS/ECCS$ schemes for sparse array A', the size of array *R* is $n+1$ and n^2+1, respectively. The size of arrays *CK* and *V* both are Sn^3. Therefore, the space complexities of the $ECRS/ECCS$ schemes are $(Sn^3+n+1)\alpha+Sn^3\beta$ and $(Sn^3+n^2+1)\alpha+Sn^3\beta$, respectively. Table 2 lists the space complexities of the CRS/CCS and the $ECRS/ECCS$ schemes. In Table 2, if $S_{CRS/CCS} > S_{ECRS/ECCS}$ is satisfied, the space complexities of the $ECRS/ECCS$ schemes are less than those of the CRS/CCS schemes, where $S_{CRS/CCS}$ and $S_{ECRS/ECCS}$ are the memory space required by the CRS/CCS and the $ECRS/ECCS$ schemes, respectively. The improved rate of space (*IRS*) is defined as $IRS(\%) = ((S_{CRS/CCS} - S_{ECRS/ECCS})/S_{CRS/CCS}) \times 100$. In general, the conditions shown in Table 2 can be satisfied easily since the size of most of sparse arrays in practical applications is large.

One of the goals to use the data compression scheme is to reduce the memory space required for sparse array operations. From Table 2, we can derive the range of usability of the CRS/CCS and the $ECRS/ECCS$ schemes according to the sparse ratio *S*. The results are shown in Table 3. In Table 3, we can see that the range of usability of the $ECRS/ECCS$ schemes is wider than that of the CRS/CCS schemes. The $ECRS/ECCS$ schemes are more suitable for practical applications with a higher sparse ratio *S* than the CRS/CCS schemes.

4 Experimental Results

In this section, we compare the compressing time of sparse arrays and the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* based on the CRS/CCS and the $ECRS/ECCS$ schemes. Due to the page limitation, we only use three-dimensional sparse arrays as test samples. All programs were implemented in *C* and were executed on an IBM RS/6000 workstation.

4.1 The Compressing Time of the *CRS/CCS* and the *ECRS/ECCS* Schemes

Assume that sparse array $A[k][i][j]$ based on the *TMR*(3) is given. If we compress sparse array *A* by using the *CRS* (*CCS*) scheme, we first compress all of the nonzero array elements along the *i* index (*j* index for *CCS*) of the sparse array. Then, we compress the nonzero array elements along the *k* index or *j* index (*k* index or *i* index for *CCS*) of the sparse array. Therefore, there are two ways, *IJK* and *IKJ* (*JIK* and *IKJ* for *CCS*), to compress sparse array *A* in the *CRS* (*CCS*) scheme. However, there is only one way in the *ECRS*/*ECCS* schemes. Table 4 shows the compressing time of the *CRS* and the *ECRS* schemes. In Table 4, we also list the *IRS* and the *IRC* for the *ECRS* scheme. From Table 4, for the *IRS*, we have two observations. First, the memory space required by the *ECRS* scheme is less than that by the *CRS* scheme since the condition $S > 0$ shown in Table 2 is satisfied. On an IBM RS/6000 workstation, an integer and a floating-point both are 4-byte long. The *IRS* can be calculated according to Table 2. Second, the *IRS* of sparse array $10 \times 10 \times 10$ with sparse ratio 0.001 is far less than that of others. The reason is that the number of nonzero array elements is too small (only 1). Therefore, the value of $S_{CRS} - S_{ECRS}$ is small.

In Table 4, for the *IRC*, we also have two observations. First, the compressing time of the *ECRS* scheme is less than that of the *CRS* scheme. The reason is that the number of one-dimensional arrays used in the *ECRS* scheme is less than that used in the *CRS* scheme. Second, the *IRC* shown in Table 4 are larger than those shown in Table 1. For example, for sparse array $10 \times 10 \times 10$ with sparse ratio 0.1, the *IRC* shown in Table 1 and Table 4 is 7.142 percent and 19.206 percent, respectively. The reason is that the cost of scanning

TABLE 4
The Compressing Time of the *CRS* and the *ECRS* Schemes

Schemes		*CRS*		*ECRS*	*Improved Rate* (%)	
Sparse Ratios	*Array Sizes*	*IJK*	*IKJ*		*IRS*	*IRC* (*average rate*)
0.1	10×10×10	0.176	0.178	0.143	32.154	19.206
	100×100×100	180.088	177.635	146.23	33.322	18.240
	200×200×200	1442.718	1432.797	1173.592	33.330	18.372
0.01	10×10×10	0.156	0.158	0.128	24.390	18.468
	100×100×100	158.273	157.734	131.781	33.221	16.595
	200×200×200	1266.479	1264.874	1043.532	33.305	17.551
0.001	10×10×10	0.157	0.156	0.128	7.14	18.210
	100×100×100	155.174	154.371	124.607	32.247	19.489
	200×200×200	1242.497	1240.752	1032.02	33.056	16.881

Time: *millisecond*

TABLE 5
The Compressing Time of the *CCS* and the *ECCS* Schemes

Schemes		*CCS*		*ECCS*	*Improved Rate* (%)	
Sparse Ratios	*Array Sizes*	*JIK*	*JKI*		*IRS*	*IRC* (*average rate*)
0.1	10×10×10	0.195	0.197	0.159	3.21	18.659
	100×100×100	333.68	318.25	148.371	30.023	54.457
	200×200×200	2847.594	2615.103	1199.344	31.672	56.009
0.01	10×10×10	0.176	0.178	0.146	×	17.608
	100×100×100	309.988	295.739	140.51	0.332	53.580
	200×200×200	2685.472	2435.533	1061.865	16.735	58.430
0.001	10×10×10	0.176	0.178	0.144	×	18.645
	100×100×100	308.022	294.132	135.681	×	54.910
	200×200×200	2657.696	2426.895	1045.681	×	58.783

Time: *millisecond*

TABLE 6
The Execution Time of *Matrix-Matrix Addition* by Compressing One Sparse Array

Schemes		*CRS*		*ECRS*	*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*IJK*	*IKJ*		*JIK*	*JKI*	
0.1	10^3	0.027	0.027	0.025	0.028	0.028	0.028
	100^3	26.599	26.166	21.797	49.709	51.390	31.577
	200^3	219.133	217.494	174.875	441.520	433.586	306.398
0.01	10^3	0.007	0.007	0.007	0.008	0.008	0.008
	100^3	4.447	3.907	3.762	4.961	5.560	4.001
	200^3	33.392	32.886	28.305	52.721	51.688	40.222
0.001	10^3	0.006	0.006	0.006	0.006	0.006	0.006
	100^3	0.492	0.484	0.479	0.486	0.552	0.422
	200^3	4.043	3.985	3.879	5.245	5.114	4.568

Time: *millisecond*

entire sparse array by the *ECRS* scheme is less than that by the *CRS* scheme, that is, the cost of scanning entire sparse array is machine-dependent [10]. However, in Table 1, we assume that the cost of scanning the entire sparse array by the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes is the same in order to simplify the analysis.

Table 5 shows the compressing time in the *CCS* and the *ECCS* schemes. In Table 5, we also list the *IRS* and the *IRC* for the *ECCS* scheme. From Table 5, for the *IRS*, we can see that the memory space required by the *ECCS* scheme is not always less than that by the *CCS* scheme. The reason is that the condition $S > 1/n$ shown in Table 2 is not always satisfied. For example, for sparse array $200 \times 200 \times 200$ with sparse ratio 0.001, the condition $S > 0.005$ is not satisfied. In Table 5, for the *IRC*, we have similar observations as those of Table 4.

From Table 4 and Table 5, we first can see that the compressing time of the *ECCS* (*CCS*) scheme is larger than that of the *ECRS* (*CRS*) scheme. The reason is that the cost of scanning entire sparse array by the *ECCS* (*CCS*) scheme is larger than that by the *ECRS* (*CRS*) scheme since all programs were implemented in the row-major data layout. Second, for some sparse arrays, the *IRC* of the *ECCS* scheme is much larger than that of the *ECRS* scheme. For example, for sparse array $100 \times 100 \times 100$ with sparse ratio 0.1, the *IRC* of the *ECCS*/*ECRS* schemes is 18.240 percent and 54.457 percent, respectively. The reason is that the costs of scanning entire sparse array for the *CCS* and the *ECCS* schemes have greater effect on overall compressing time than those of the *CRS* and the *ECRS* schemes.

4.2 The Execution Time of Sparse Array Operations Based on the *CRS/CCS* and the *ECRS/ECCS* Schemes

Tables 6 and 7 show the execution time of *matrix-matrix addition* based on the *CRS/CCS* and the *ECRS/ECCS* schemes by compressing one and two sparse arrays, respectively. For the case where one sparse array is compressed, we use the indices of array elements in the compressed sparse array to find the corresponding array elements in the noncompressed sparse array before performing addition operations. For the case where two sparse arrays are compressed, we need to check if the indices of array

TABLE 7
The Execution Time of *Matrix-Matrix Addition* by Compressing Two Sparse Arrays

Schemes		*CRS*		*ECRS*	*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*IJK*	*IKJ*		*JIK*	*JKI*	
0.1	10^3	0.096	0.087	0.07	0.135	0.118	0.105
	100^3	106.702	91.38	66.03	110.376	106.695	67.540
	200^3	873.434	828.625	524.72	897.346	868.475	609.974
0.01	10^3	0.017	0.015	0.014	0.023	0.021	0.017
	100^3	9.178	8.816	7.838	11.911	11.169	8.445
	200^3	78.243	73.842	55.5	80.763	78.935	65.247
0.001	10^3	0.009	0.009	0.009	0.012	0.012	0.012
	100^3	0.794	0.775	0.718	1.040	0.911	0.837
	200^3	6.723	6.655	6.598	7.078	6.918	6.514

Time: *millisecond*

TABLE 8
The Execution Time of *Matrix-Matrix Multiplication* by Compressing One Sparse Array

Schemes		*CRS*		*ECRS*	*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*IJK*	*IKJ*		*JIK*	*JKI*	
0.1	10^3	0.296	0.295	0.276	0.234	0.232	0.224
	100^3	3597.63	3335.90	3096.54	2600.03	2578.90	2316.29
	200^3	161984	150129	138897	141779	141029	126921
0.01	10^3	0.037	0.035	0.035	0.029	0.029	0.029
	100^3	455.15	419.87	395.51	265.96	256.44	231.65
	200^3	41274	40388	38301	41645	41476	36891
0.001	10^3	0.009	0.009	0.009	0.007	0.007	0.007
	100^3	44.55	44.10	43.10	24.81	24.53	22.42
	200^3	729.42	722.76	712.61	407.33	409.07	369.49

Time: *millisecond*

elements of a compressed sparse array are the same as those in another compressed sparse array before performing addition operations.

From Tables 6 and 7, we can see that the execution time of *matrix-matrix addition* based on the *ECRS/ECCS* schemes is less than that based on the *CRS/CCS* schemes. For Table 6, the reason is that the cost of indirect data access in the *ECRS/ECCS* schemes is less than that in the *CRS/CCS* schemes. For Table 7, the reason is that the cost of index comparisons in the *ECRS/ECCS* schemes is less than that in the *CRS/CCS* schemes.

Table 8 shows the execution time of *matrix-matrix multiplication* based on the *CRS/CCS* and the *ECRS/ECCS* schemes by compressing one sparse array. From Table 8, for the execution time of *matrix-matrix multiplication*, we have similar observations as those of Table 6.

5 CONCLUSIONS

In this paper, we have presented the *ECRS/ECCS* data compression schemes for multidimensional sparse arrays based on the *EKMR* scheme. From the theoretical analysis and experimental results, we have the following conclusions:

1. The time complexity for compressing a multidimensional sparse array based on the *ECRS/ECCS* schemes is less than that based on the *CRS/CCS* schemes.
2. For most of the sparse arrays in practical applications, the space complexity of compressing a multidimensional sparse array based on the *ECRS/ECCS* schemes is less than that based on the *CRS/CCS* schemes.
3. The range of usability of the *ECRS/ECCS* schemes is wider than that of the *CRS/CCS* schemes for practical applications.
4. The performance of sparse array operations based on the *ECRS/ECCS* schemes is better than that based on the *CRS/CCS* schemes.

ACKNOWLEDGMENTS

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

REFERENCES

[1] G. Bandera, P.P. Trabado, and E.L. Zapata, "Local Enumeration Techniques for Sparse Algorithms," *Proc. Int'l Symp. Parallel Processing,* pp. 52-56, Apr. 1998.
[2] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van, *Templates for the Solution of Linear Systems: Building Blocks for the Iterative Methods,* second ed. 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, pp. 305-339, Mar. 2001.
[4] J.K. Cullum and R.A. Willoughby, *Lanczos Algorithms for Large Symmetric Eignenvalue Computations.* Boston: Birkhauser, 1985.
[5] I. Duff, R. Grimes, and J. Lewis, "User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)," Technical Report RAL 92-086, Rutherford Appleton Laboratory, 1992.
[6] 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. Euromicro Conf.,* pp. 345-348, Aug. 1998.
[7] G.H. Golub and C.F. Van Loan, *Matrix Computations,* second ed. Baltimore, Md.: The John Hopkins Univ. Press, 1989.
[8] C.W. Kebler and C.H. Smith, "The SPARAMAT Approach to Automatic Comprehension of Sparse Matrix Computations," *Proc. Int'l Workshop Program Comprehension,* pp. 200-207, May 1999.
[9] V. Kotlyar, K. Pingali, and P. Stodghill, "Compiling Parallel Code for Sparse Matrix Applications," *Proc. Supercomputing Conf.,* pp. 20-38, Aug. 1997.
[10] 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, pp. 327-345, Mar. 2002.

[11] P.D. Sulatycke and K. Ghose, "Caching Efficient Multithreaded Fast Multiplication of Sparse Matrices," *Proc. Merged Int'l Symp. Parallel Processing and Parallel and Distributed Processing,* pp. 117-124, 1998.

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

[13] 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, pp. 1068-1083, Oct. 1997.

[14] J.B. White and P. Sadayappan, "On Improving the Performance of Sparse Matrix-Vector Multiplication," *Proc. Int'l Conf. High Performance Computing,* pp. 711-725, 1997.

[15] E.L. Zapata, O. Plata, R. Asenjo, and G.P. Trabado, "Data-Parallel Support for Numerical Irregular Problems," *J. Parallel Computing,* vol. 25, nos. 13-14, pp. 1971-1944, 1999.

[16] 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,* pp. 313-322, July 1994.