Efficient Data Compression Methods for Multi-Dimensional Sparse Array Operations

Chun-Yuan Lin, Yeh-Ching Chung, and Jen-Shiuh Liu
Department of Information Engineering
Feng Chia University, Taichung, Taiwan 407, *ROC*
Email: {cylin, ychung, liuj}@iecs.fcu.edu.tw

Abstract

For sparse array operations, in general, the sparse arrays are compressed by some data compression schemes in order to obtain better performance. The Compressed Row/Column Storage (CRS/CCS) schemes are the two common used data compression schemes for sparse arrays in the traditional matrix representation (TMR). When extended to higher dimensional sparse arrays, array operations used the CRS/CCS schemes usually do not perform well. In this paper, we propose two data compression schemes, extended Karnaugh map representation-Compressed Row/Column Storage (ECRS/ ECCS) for multi-dimensional sparse arrays based on the EKMR scheme. To evaluate the proposed schemes, both theoretical analysis and experimental test are conducted. In theoretical analysis, we analyze CRS/CCS and ECRS/ECCS schemes in terms of the time complexity, the space complexity, and the range of their usability for practical applications. In experimental test, we compare the performance of matrix-matrix addition and matrix-matrix multiplication sparse array operations that use the CRS/CCS and ECRS/ECCS schemes. The experimental results show that sparse array operations based on the ECRS/ECCS schemes outperform those based on the CRS/CCS schemes for all test samples.

Index Terms－Data compression scheme, Sparse array operation, Multi-dimensional sparse array, Karnaugh Map.

1. Introduction

Array operations are useful in a large number of important scientific codes, such as molecular dynamics [5], finite-element methods [8], climate modeling [16], etc. For sparse array operations, in general, the sparse arrays are compressed by some data compression schemes in order to obtain better performance. Many data compression schemes have been proposed, such as *Compressed Column Storage* (*CCS*) [1], *Compressed Row Storage* (*CRS*) [1], *Jagged Diagonal format* (*JAD*) [1], and *Symmetric Sparse Skyline format* (*SSS*) [1], etc. Among them, the *CRS*/*CCS* schemes are the two common used data compression schemes due to their simplicity and pure with weak dependence relationship between array elements in a sparse array.

A multi-dimensional array can be viewed as a collection of the two-dimensional arrays. For example, one can use 5 separate 4×3 two-dimensional arrays to represent a three-dimensional array of size 5×4×3. This scheme is called *traditional matrix representation* (*TMR*) that is also known as *canonical data layouts* [4]. The compression schemes mentioned above are based on the *TMR* scheme. For the *CRS*/*CCS* schemes, a two-dimensional sparse array based on the *TMR* scheme can be compressed into three one-dimensional arrays. Therefore, for a sparse array operation, we only operate these non-zero array elements to obtain better performance and use less memory space. However, for higher dimensional sparse arrays, array operations based on *CRS*/*CCS* schemes usually do not perform well. The reasons are two-fold. First, the number of one-dimensional arrays increases as the dimension increases because more one-dimensional arrays are needed to store extra indices of non-zero array elements for higher dimensional sparse array. This increases the time and the memory space of compressing a multi-dimensional sparse array. Second, the costs of indirect data access and index comparisons for multi-dimensional sparse array operations increase as the dimension increases.

In our previous work [13-14], we have proposed a new scheme called *extended Karnaugh map representation* (*EKMR*) for the multi-dimensional array representation. This scheme is suitable for the multi-dimensional dense or sparse array without using the data compression scheme.

In this paper, we propose two new data compression schemes, *extended Karnaugh map representation-Compressed Row*/*Column Storage* (*ECRS*/*ECCS*) for

Proceedings of the First International Symposium on Cyber Worlds (CW'02)
0-7695-1862-1/02 $17.00 © 2002 **IEEE**

multi-dimensional sparse array 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). If k = 3 or 4, the *ECRS*/*ECCS* schemes use two one-dimensional integer arrays and one one-dimensional floating-point array to compress the sparse array. If $k > 4$, the *ECRS*/*ECCS* schemes first use two one-dimensional integer arrays and an one-dimensional floating-point array to compress the 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 one-dimensional arrays of each *EKMR*(4).

To evaluate the proposed schemes, both theoretical analysis and experimental test are conducted. In theoretical analysis, we analyze *CRS*/*CCS* and *ECRS*/*ECCS* schemes in terms of the time complexity, the space complexity, and the range of their usability for practical applications. From the theoretical analysis, we can see that the time and the space complexities for compressing a multi-dimensional sparse array based on the *ECRS*/*ECCS* schemes are less than those based on 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 test, we compare the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* sparse array operations for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes. The experimental results show that sparse array operations based on the *ECRS*/*ECCS* schemes outperform those based on the *CRS*/*CCS* schemes. There are two reasons. First, for the *ECRS*/*ECCS* schemes, the number of one-dimensional arrays does not increase as the dimension increases since a multi-dimensional sparse array based on the *EKMR* scheme is represented by a set of two-dimensional sparse array. The time and the memory space required to compress a sparse array can be reduced. Second, the costs to perform the indirect data access and index comparisons of sparse array operations for the *ECRS*/*ECCS* schemes are less than those of the *CRS*/*CCS* schemes.

This paper is organized as follows. In Section 2, a brief survey of related work will be presented. Section 3 will describe the *ECRS*/*ECCS* schemes and analyze their theoretical performance along with the *CRS*/*CCS* schemes for compressing a multi-dimensional sparse array. The efficient algorithms of multi-dimensional sparse array operations based on the *ECRS*/*ECCS* schemes will be given in Section 4. The performance comparisons of these algorithms will be given in Section 5.

2. Related Work

Many methods for improving sparse array computation have been proposed in the literature. We briefly describe the related researches. Kotlyar *et al.* [11-12] presented a relational algebra based framework for compiling efficient sparse array code from dense DO-Any loops and a specified sparse array. Sularycke and Ghose [15] showed a simple sequential loop interchange algorithm that can produce a better performance than existing algorithms for sparse array multiplication. Zapata *et al.* [6-7] analyzed the cache effects for the array operations. They established the cache probabilistic modeling and modeled the cache behavior for sparse array operations. Kebler and Smith [10] described a system, SPARAMAT, for concept comprehension that is particularly suitable for sparse array codes. Lee *et al*. [2-3] presented an efficient library for parallel sparse computations with Fortran 90 array intrinsic operations. They provide a new data compression scheme, which is obtained by extending the *CRS*/*CCS* schemes for two-dimensional sparse arrays, for multi-dimensional sparse arrays based on the *TMR* scheme. Kandemir *et al.* [9] proposed a compiler technique to perform loop and data layout transformations to solve the global optimization problem on sequential and multiprocessor machines. They used one data layout for the entire program and improved the performance by using the loop transformation scheme. However, their method may be difficult to extend to sparse array programs. The reason is that sparse array programs, in general, use the data compression scheme, which heavily use of indirect addressing through index stored in index arrays. Since these index arrays are read at run-time, compiler cannot analyze which non-zero array element will actually be accessed in a given loop.

3. The *ECRS*/*ECCS* Schemes

Before presenting the *ECRS*/*ECCS* schemes, we briefly describe the *EKMR* scheme for three-dimensional arrays. Details of the *EKMR* scheme for four- or higher dimensional arrays can be found in [13]. In the following, we describe the *EKMR* scheme based on the row-major storage scheme, such as *C* language. The idea of the *EKMR* scheme is based on the Karnaugh map. Let $A[k][i][j]$ denote a three-dimensional array in the *TMR*(3). The corresponding *EKMR*(3) of array $A[3][4][5]$, is shown in Figure 1. The *EKMR*(3) is represented by a two-dimensional array with the size of 4×(3×5).

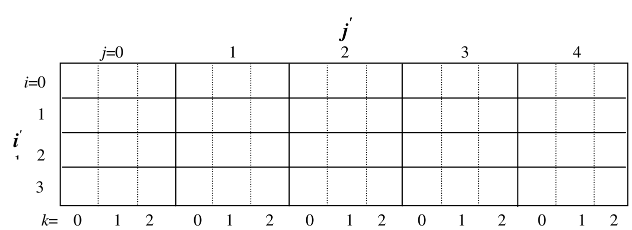

Figure 1: The *EKMR*(3) scheme.

Proceedings of the First International Symposium on Cyber Worlds (CW'02)
0-7695-1862-1/02 $17.00 © 2002 IEEE

The difference between the $TMR(3)$ and the $EKMR(3)$ is the placement of array elements along the direction indexed by k. 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 i, whereas the index j' is a combination of the indices j and k. A more concrete example is given in Figure 2.

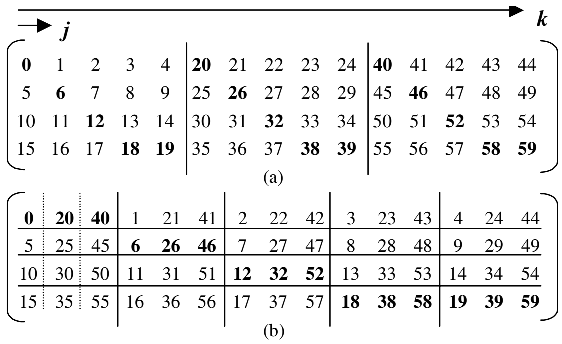

Figure 2: (a) A three-dimensional array in the *TMR*(3). (b) The corresponding *EKMR*(3).

3.1 The *CRS*/*CCS* Schemes

Given a two-dimensional sparse array based on the $TMR(2)$, the *CRS* (*CCS*) scheme using two one-dimensional integer arrays, *RO* and *CO*, and an one-dimensional floating-point array, *VL*, to compress all of non-zero array elements along rows (columns for *CCS*) of the sparse array. Array *RO* stores information of non-zero array elements of each row (column for *CCS*). The number of non-zero array elements in the ith row (jth column for *CCS*) can be obtained by subtracting the value of $RO[i]$ from $RO[i+1]$. Array *CO* stores the column (row for *CCS*) indices of non-zero array elements of each row (column for *CCS*). Array *VL* stores the values of non-zero array elements of the sparse array. The base of these three arrays is 0. An example of the *CRS*/*CCS* schemes for a two-dimensional sparse array based on the $TMR(2)$ is given in Figure 3. Figure 3(a) shows a 3×4 sparse array A with 6 non-zero array elements. Figures 3(b) and 3(C) show the *CRS*/*CCS* schemes for the sparse array, respectively. In Figure 3(b), the number of non-zero array elements in the second row can be obtained by $RO_{CRS}[3] - RO_{CRS}[2] = 7 - 5 = 2$. The column indices of non-zero array elements of the second row are stored in $CO_{CRS}[RO_{CRS}[2]-1]$, …, $CO_{CRS}[RO_{CRS}[3]-2]$. The non-zero array elements of the second row are stored in $VL_{CRS}[4:5]$.

Based on the *CRS*/*CCS* schemes for the two-dimensional sparse array, for a three-dimensional sparse array based on the $TMR(3)$, it also can be compressed by the *CRS*/*CCS* schemes by adding one one-dimensional integer array, *KO*. Array *KO* stores the indices of all non-zero array elements in the third dimension of the sparse array. For a four- or higher dimensional sparse array based on the *TMR* scheme, more one-dimensional integer arrays are added to store indices of all non-zero array elements in the fourth or higher dimension. An example of the *CRS*/*CCS* schemes for a three-dimensional sparse array based on the $TMR(3)$ is shown in Figure 4.

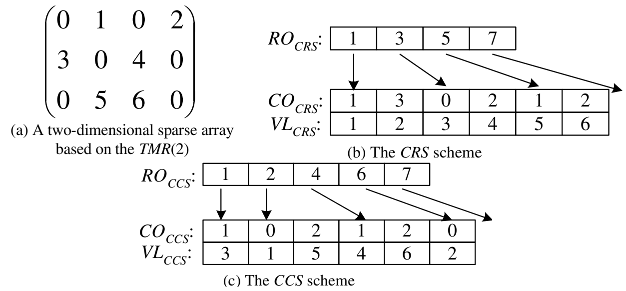

Figure 3: The *CRS*/*CCS* schemes for a two-dimensional sparse array based on the *TMR*(2).

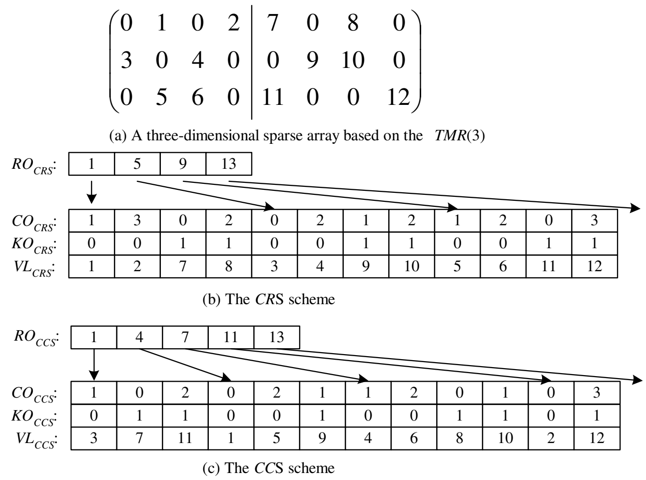

Figure 4: The *CRS*/*CCS* schemes for a three-dimensional sparse array based on the *TMR*(3).

3.2 The *ECRS*/*ECCS* Schemes

The main idea of the *EKMR* scheme is to represent a multi-dimensional array by a set of two-dimensional arrays. Therefore, the *ECRS*/*ECCS* schemes use a set of two one-dimensional integer arrays, R and CK, and one one-dimensional floating-point array, V, to compress a multi-dimensional sparse array.

Given a three-dimensional sparse array based on the $EKMR(3)$, the *ECRS* (*ECCS*) scheme compresses all of non-zero array elements along the rows (columns for *ECCS*) of the sparse array. Array R stores information of non-zero array elements of each row (column for *ECCS*). The number of non-zero array elements in the ith row (jth 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 non-zero array elements of each row (column for *ECCS*). Array V stores the values

Proceedings of the First International Symposium on Cyber Worlds (CW'02)
0-7695-1862-1/02 $17.00 © 2002 IEEE

of non-zero array elements of the sparse array. The base of these three arrays is 0. An example of the *ECRS*/*ECCS* schemes for a three-dimensional sparse array based on the *EKMR*(3) is given in Figure 5. Figure 5(a) shows a 3×8 sparse array A' with 12 non-zero array elements based on the *EKMR*(3) whose *TMR*(3) is shown in Figure 4(a). Figures 5(b) and 5(c) show the *ECRS*/*ECCS* schemes of the sparse array, respectively. By the definition of the *EKMR* scheme, a four-dimensional sparse array based on the *EKMR*(4) is also presented by a two-dimensional sparse array. We can use two one-dimensional integer arrays and one one-dimensional floating-point array to compress the four-dimensional sparse array.

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

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

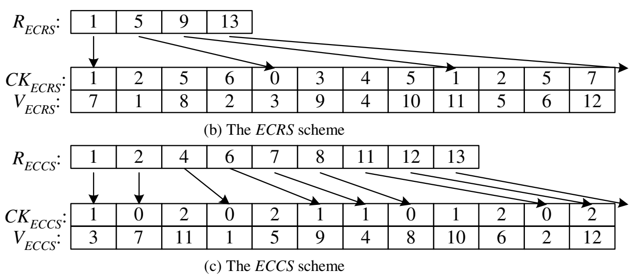

Figure 5: The *ECRS*/*ECCS* schemes for a three-dimensional sparse array based on the *EKMR*(3).

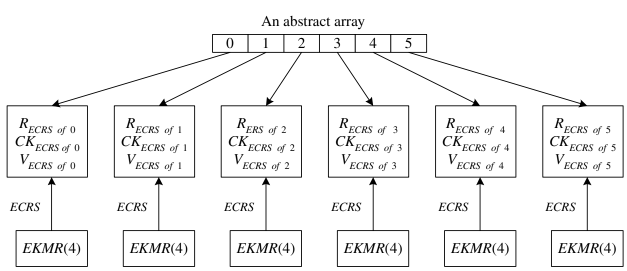

Figure 6: The *ECRS* scheme for a six-dimensional sparse array based on the *EKMR*(6).

Given a k-dimensional sparse array with a size of m along each dimension based on the *EKMR*(k), the *EKMR*(k) can be represented by m^{k-4} *EKMR*(4). Since we use an abstract one-dimensional array to link all the sparse arrays in the *EKMR*(4), to compress this k- dimensional sparse array, the *ECRS*/*ECCS* schemes first compress each *EKMR*(4) sparse array by using two one-dimensional integer arrays and one one-dimensional floating-point array. Then, an abstract array with a size of m^{k-4} is used to link these three one-dimensional arrays of each *EKMR*(4). For example, assume that there is a six-dimensional sparse array A with a size of 3×2×2×3× 4×5 in the *TMR*(6). The array A' in the *EKMR*(6) can be represented by six (3×2) arrays in the *EKMR*(4) with a size of (2×4)×(3×5). If we compress the sparse array based on the *ECRS* scheme, we first compress each *EKMR*(4) to three one-dimensional arrays, *R*, *CK*, and *V*. Then, we use an abstract array with a size of 6 to link these three one-dimensional arrays of each *EKMR*(4). An example is shown in Figure 6.

3.3 Theoretical Analysis

In the following, we analyze the theoretical performance for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes in terms of the time complexity, the space complexity, and the range of their usability for practical applications. In the following analysis, we assume that a k-dimensional sparse array A has n^k array elements and the sparse probability [7] for each array element is equal.

First, we analyze the time complexity for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes. Assume that a three-dimensional sparse array A based on the *TMR*(3) with size $n \times n \times n$ is given and the sparse ratio of A is S. The number of non-zero elements of array A is Sn^3. Let sparse array A' be the corresponding array A in the *EKMR*(3). In the *CRS*/*CCS* schemes, four one-dimensional arrays, *RO*, *CO*, *KO*, and *VL* are used for compression. They first need to scan entire array A to find all of non-zero array elements. Then, they need to record the information of each non-zero array element to these four arrays. Therefore, the time complexity for the *CRS*/*CCS* schemes is n^3+4Sn^3. Similarly, if we compress array A' with the *ECRS*/*ECCS* schemes by using three one-dimensional arrays, *R*, *CK*, and *V*, the time complexity for the *ECRS*/*ECCS* schemes is n^3+3Sn^3. For four- or higher dimensional sparse arrays, we can obtain the time complexities of *CRS*/*CCS* and *ECRS*/*ECCS* schemes in a similar manner. Table 1 lists the time complexities for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes for $k \geq 2$. In Table 1, the improved rate is defined as follows: *Improved Rate* (%) = $\frac{Time(CRS/CCS) - Time(ECRS/ECCS)}{Time(CRS/CCS)} \times 100$, where *Time*(*CRS*/*CCS*) and *Time*(*ECRS*/*ECCS*) are the time required by *CRS*/*CCS* and *ECRS*/*ECCS* schemes to perform a compression, respectively. From Table 1, we can see that the improved rates increase as the dimension increases. In the *CRS*/*CCS* schemes, array A is compressed by using three one-dimensional integer arrays, *RO*, *CO*, and *KO*, and one one-dimensional floating-point array, *VL*. The size of *RO* is $n+1$, the size of *CO*, *KO*, and *VL* arrays are all Sn^3. Assume that an integer is α bytes long and a floating-point is β bytes long. The space complexity of the *CRS*/*CCS* schemes for A is $(2Sn^3+n+1)\alpha + Sn^3\beta$. Similarly, the space complexities of the *ECRS*/*ECCS* schemes for A' are $(Sn^3+n+1)\alpha + Sn^3\beta$ and $(Sn^3+n^2+1)\alpha + Sn^3\beta$, respectively. Note that for the *ECCS* scheme based on the *EKMR*(3), the size of array *R* is n^2+1.

Table 2 lists the theoretical analysis of the space complexities for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes. If the conditions listed in Table 2 are satisfied, the space complexity of the *ECRS*/*ECCS* schemes is less than that of the *CRS*/*CCS* schemes. In general, the size of sparse array is large and the conditions can be easily satisfied.

Finally, we discuss the range of their usability for practical applications. One of goal to use the data compression scheme is to decrease the memory space requirement. Therefore, the space complexity for the non-compressed sparse array must larger than that of the compressed sparse array. From Table 2, we can derive the range of usability of the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes for practical application according to the sparse ratio *S*. The results are shown in Table 3. From Table 3, we can see that the range of usability of the *CRS*/*CCS* schemes reduces as the dimension increases. Hence, the range of usability of the *ECRS*/*ECCS* schemes is wider than that of the *CRS*/*CCS* schemes for practical applications. The *ECRS*/*ECCS* schemes are more suitable for practical applications with a higher sparse ratio than the *CRS*/*CCS* schemes.

Table 1: Time complexities.

Dimensions \ Schemes	*CRS*/*CCS*	*ECRS*/*ECCS*	*Improved Rate* (%)
3-*D*	n^3+4Sn^3	n^3+3Sn^3	$\frac{S}{1+4S}\times 100$
4-*D*	n^4+5Sn^4	n^4+3Sn^4	$\frac{2S}{1+5S}\times 100$
k-*D* ($k \geq 2$)	n^k+(k+1)Sn^k	n^k+3Sn^k	$\frac{(k-2)S}{1+(k+1)S}\times 100$

Table 2: Space complexities.

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

* *Condition* : *Space*(*CRS*/*CCS*) > *Space*(*ECRS*/*ECCS*).

Table 3: The range of usability.

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

4. Algorithms for Sparse Array Operations

Most algorithms of sparse array operations are based on the *CRS*/*CCS* schemes. The structure of compressing a sparse array based on the *ECRS*/*ECCS* schemes is quite different from that based on the *CRS*/*CCS* schemes. Hence, we need to redesign algorithms for sparse array operations with the *ECRS*/*ECCS* schemes. For the page limitation, in this section, we only present efficient algorithms for *matrix-matrix addition* and *matrix-matrix multiplication* sparse array operations based on the *ECRS*(3)/*ECCS*(3) schemes. For both sparse array operations, we can consider the compression of one or two sparse arrays. However, the compression of two sparse arrays for both sparse array operations is complicated and has many issues for discussions. For simplicity, in this paper, we only consider the compression of one sparse array. However, we do give some experimental results for the case where two sparse arrays are compressed in Section 6. For the algorithms based on the *CRS*/*CCS* schemes, please refer to [1].

4.1 Matrix-Matrix Addition Algorithms

Assume that *A* and *B* are two $n \times n \times n$ three-dimensional sparse arrays in the *TMR*(3). Let A' and B' be the corresponding arrays of *A* and *B* in the *EKMR*(3), respectively. According to the *ECRS*/*ECCS* schemes, array A' can be compressed into three one-dimensional arrays, *R*, *CK*, and *V*. Based on the *ECRS*/*ECCS* schemes, the efficient algorithms for $B' = A' + B'$ are given below.

Algorithm matrix-matrix_addition_ECRS_EKMR(3)
1. *for* ($i = 0$; $i < n$; i++)
2. *for* ($j = R_{ECRS}[i]$; $j < R_{ECRS}[i+1]$; j++)
3. $B'[i][CK_{ECRS}[j-1]] = V_{ECRS}[j-1] + B'[i][CK_{ECRS}[j-1]]$;

end_of_matrix-matrix_addition_ECRS_EKMR(3)

Algorithm matrix-matrix_addition_ECCS_EKMR(3)
1. *for* ($i = 0$; $i < n$; i++)
2. *for* ($j = R_{ECCS}[i]$; $j < R_{ECCS}[i+1]$; j++)
3. $B'[CK_{ECCS}[j-1]][i] = V_{ECCS}[j-1] + B'[CK_{ECCS}[j-1]][i]$;

end_of_matrix-matrix_addition_ECCS_EKMR(3)

4.2 Matrix-Matrix Multiplication Algorithms

Assume that *A* and *B* are two $n \times n \times n$ three-dimensional sparse arrays with the sparse ratio *S* in the *TMR*(3). Let A' and B' be the corresponding arrays of *A* and *B* in the *EKMR*(3), respectively. For the *ECRS*/*ECCS* schemes, array A' can be compressed into three one-dimensional arrays, *R*, *CK*, and *V*. The algorithms for $C' = A' \times B'$ based on the *ECRS*/*ECCS* schemes are given below.

```
Algorithm matrix-matrix_multiplication_ECRS_EKMR(3)
   1. for ( i = 0 ; i < Sn^3 ; i++)
   2.    K[i]=CK_ECRS[i] / n ;
   3.    CK_ECRS[i]= CK_ECRS[i] % n ;
   4 for ( i = 0 ; i < n ; i++)
   5.       for (k = 0; k < n; k++)
   6.          r1 = k × n;
   7.          for (j = R_ECRS[i]; j< R_ECRS[i+1]; j++)
   8.             r2 = CK_ECRS[j-1] + r1;
   9.             C'[i][r2] = C'[i][r2] + V_ECRS[j-1] × B'[K[j-1]][r2];
end_of_matrix-matrix_multiplication_ECRS_EKMR(3)
```

```
Algorithm matrix-matrix_multiplication_ECCS_EKMR(3)
   1. for ( i = 0 ; i < n^2 ; i++)
   2.    for (j = R_ECCS[i]; j< R_ECCS[i+1]; j++)
   3.       r3 = i % n ;
   4.       r4 = i / n ;
   5.       r6 = CK_ECCS[j-1];
   6.       for (k = 0; k < n; k++)
   7.          r5 = k × n + r3;
   8.          C'[r6][r5] = C'[r6][r5] + V_ECCS[j-1] × B'[r4][r5];
end_of_matrix-matrix_multiplication_ECCS_EKMR(3)
```

5. Experimental Results

To evaluate the performance of the proposed data compression schemes, we compare the compression time of the *ECRS*/*ECCS* and the *CRS*/*CCS* schemes. We also compare the execution time of sparse array operations based on the *ECRS*/*ECCS* and the *CRS*/*CCS* schemes. For the sparse array operations, *matrix-matrix addition* and *matrix-matrix multiplication* operations are implemented. For all the implemented sparse array operations, we use three-dimensional arrays as test samples. The compression algorithms and the sparse array operations were implemented in *C* and were executed on an IBM RS/6000 workstation.

5.1 The Compressing Time

For a *k*-dimensional sparse array based on the *TMR*(*k*) where $k \geq 2$, there are (*k*-1)! ways for the *CRS*/*CCS* schemes to compress the sparse array. Assume that a three-dimensional sparse array *A*[*k*][*i*][*j*] based on the *TMR*(3) is given. If we compress array *A* by using the *CRS* (*CCS*) scheme, we first compress all of non-zero array elements along *i* index (*j* index for *CCS*) of the sparse array. Then, we compress non-zero array elements along *k* or *j* index (*k* or *i* index for *CCS*). Therefore, there are two ways *IJK* and *IKJ* (*JIK* and *IKJ* for *CCS*) to compress array *A* in the *CRS* (*CCS*) scheme.

For a *k*-dimensional sparse array based on the *EKMR*(*k*), where $k \geq 3$, there is only one way for the *ECRS*/*ECCS* schemes to compress the sparse array. Table 4 shows the execution time for compressing three-dimensional sparse arrays with various sparse ratios and array sizes based on the *CRS* and the *ECRS* schemes. From Table 4, we can see that the performance of compressing three-dimensional sparse arrays based on the *ECRS* scheme is better than that based on the *CRS* scheme for all test samples. The results match the theoretical analysis described in Section 4. From Table 4, we also can see that the performance of compressing spare arrays using the *IJK* order is different from that of the *IKJ* order in the *CRS* scheme. Since the size of *R* for the *ECCS* scheme is larger than the size of *RO* for the *CCS* scheme, we also list the execution time of compressing three-dimensional sparse arrays with various sparse ratios and array sizes based on the *CCS* and the *ECCS* schemes in Table 5. From Table 5, we have similar observations as those of Table 4. In Table 5, we also can see that the compression time of the *CCS* scheme is much larger than that of the *CRS* scheme. The reason is that the compression algorithms were implemented in *C*. However, for the *ECRS*/*ECCS* schemes, the difference is not that large. The reason is that the data locality of the *EKMR* scheme is better than that of the *TMR* scheme.

Table 4: The execution time of compressing three-dimensional sparse arrays.

Schemes		*CRS*		*ECRS*
Sparse Ratios	*Array Sizes*	*IJK*	*IKJ*	
0.1	10×10×10	0.176	0.178	0.143
	100×100×100	180.088	177.635	146.23
	200×200×200	1442.718	1432.797	1173.592
0.01	10×10×10	0.156	0.158	0.128
	100×100×100	158.273	157.734	131.781
	200×200×200	1266.479	1264.874	1043.532
0.001	10×10×10	0.157	0.156	0.128
	100×100×100	155.174	154.371	124.607
	200×200×200	1242.497	1240.752	1032.02

Time: *ms*

Table 5: The execution time of compressing three-dimensional sparse arrays.

Schemes		*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*JIK*	*JKI*	
0.1	10×10×10	0.176	0.176	0.159
	100×100×100	333.68	318.25	148.371
	200×200×200	2847.594	2615.103	1199.344
0.01	10×10×10	0.156	0.157	0.146
	100×100×100	309.988	295.739	140.51
	200×200×200	2685.472	2435.533	1061.865
0.001	10×10×10	0.154	0.155	0.144
	100×100×100	308.022	294.132	135.681
	200×200×200	2657.696	2426.895	1045.681

Time: *ms*

5.2 The Execution time of Sparse Array Operations

Table 6 shows the execution time of algorithms for the *matrix-matrix addition* sparse array operation based on the *CRS* and the *ECRS* schemes by compressing one three-dimensional sparse array with various sparse ratios and array sizes. In Table 6, we also compare the execution time of algorithms for the *matrix-matrix addition* sparse array operation with and without the data compression scheme. From Table 6, we can see that the performance of the *matrix-matrix addition* sparse array operation based on the *ECRS* scheme is better than that based on the *CRS* scheme. The reason is that the cost of indirect data access for the *ECRS* scheme is less than that for the *CRS* scheme. Moreover, the performance of the *matrix-matrix addition* sparse array operation with the data compression scheme is better than that without the data compression scheme.

Table 7 shows the execution time of algorithms for the *matrix-matrix addition* sparse array operation based on the *CCS* and the *ECCS* schemes by compressing one three-dimensional sparse array with various sparse ratios and array sizes. From Table 7, we have similar observations as those shown in Table 6.

Table 8 shows the execution time of algorithms for the *matrix-matrix addition* sparse array operation based on the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes by compressing two three-dimensional sparse arrays with various sparse ratios and array sizes. From Table 8, we can see that the performance of the *matrix-matrix addition* sparse array operation based on the *ECRS*/*ECCS* schemes is better than that based on the *CRS*/*CCS* schemes. The reason is that the cost of index comparison for the *ECRS*/*ECCS* schemes is less than that for the *CRS*/*CCS* schemes.

Table 9 shows the execution time of algorithms for the *matrix-matrix multiplication* sparse array operation based on the *CRS*/*CCS* and the *ECRS*/*ECCS* schemes by compressing one three-dimensional sparse array with various sparse ratios and array sizes. From Table 9, we can see that the performance of the *matrix-matrix multiplication* sparse array operation based on the *ECRS*/*ECCS* schemes is better than that based on the *CRS*/*CCS* schemes. The reason is that the cost of indirect data access in the *ECRS*/*ECCS* schemes is less than that in the *CRS*/*CCS* schemes.

Table 7: The execution time for the *matrix-matrix addition* operation by compressing one sparse array.

Schemes		*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*C-N JIK*	*C-N JKI*	*C-N*
0.1	10×10×10	0.028	0.028	.028
	100×100×100	49.709	51.390	31.577
	200×200×200	441.520	433.586	306.398
0.01	10×10×10	0.008	0.008	0.008
	100×100×100	4.961	5.560	4.001
	200×200×200	52.721	51.688	40.222
0.001	10×10×10	0.006	0.006	0.006
	100×100×100	0.486	0.552	0.422
	200×200×200	5.245	5.114	4.568

Time: *ms*

**C-N*: *Compressed array-Non-compressed array addition*.

6. Conclusions

In this paper, we have presented the *ECRS*/*ECCS* data compression schemes for multi-dimensional sparse array based on the *EKMR* scheme. We have analyzed the theoretical performance for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes in terms of the time complexity, the space complexity, and the range of their usability for practical applications. From the theoretical analysis, we can conclude that the time and the space complexities for compressing a multi-dimensional sparse array based on the *ECRS*/*ECCS* schemes are less than those based on 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 test, we also compared the execution time of *matrix-matrix addition* and *matrix-matrix multiplication* sparse array operations for both *CRS*/*CCS* and *ECRS*/*ECCS* schemes. The experimental results show that sparse array operations based on the *ECRS*/*ECCS* schemes outperform those based on the *CRS*/*CCS* schemes for all test samples. The results encourage us using the *ECRS*/*ECCS* schemes to compress multi-dimensional sparse arrays.

Acknowledgments

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

References

[1] 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.

[2] Rong-Guey Chang, Tyng-Ruey Chung, and Jenq Kuen Lee, "Compiler Optimization for Parallel Sparse Programs with Array Intrinsics of Fortran 90," *In the International Conference on Parallel Processing*, September, 1999.

[3] Rong-Guey Chang, Tyng-Ruey Chung, and Jenq Kuen Lee, "Parallel Sparse Supports for Array Intrinsic Functions of Fortran 90," accepted by Journal of Supercomputing.

[4] M. Cierniak and W. Li, "Unifying Data and Control Transformations for Distributed Shared Memory Machines," *Technical Report*, November 1994.

[5] J.K. Cullum and R.A. Willoughby, "Algorithms for Large Symmetric Eignenvalue Computations," vol. 1 , 1985.

[6] B. B. Fraguela, R. Doallo, E. L. Zapata, "Cache Misses Prediction for High Performance Sparse Algorithms, " 4th *International Euro-Par Conference*, pp.224-233, 1998.

[7] B. B. Fraguela, R. Doallo, E. L. Zapata, "Cache Probabilistic Modeling for Basic Sparse Algebra Kernels Involving Matrices with a Non-Uniform Distribution, " 24th *IEEE Euromicro Conference*, pp.345-348, August, 1998.

[8] G.H. Golub and C.F. Van Loan, *Matrix Computations,* 2nd ed. (Johns Hopkins Univ.Press, Baltimore, 1989)

[9] Mahmut Kandemir, J. Ramanujam, Alok Choudhary, "Improving Cache Locality by a Combination of Loop and Data Transformations," *IEEE Trans. on Computers*, vol. 48, no. 2, February 1999.

[10] Christoph W. Kebler and Craig H. Smith, "The SPARAMAT Approach to Automatic Comprehension of Sparse Matrix Computations," *In Proceedings of the Seventh International Workshop on Program Comprehension*, pp. 200-207, 1999.

[11] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill, "A Relation Approach to the Compilation of Sparse Matrix Programs," *In Euro Par*, August 1997.

[12] Vladimir Kotlyar, Keshav Pingali, and Paul Stodghill, "Compiling Parallel Code for Sparse Matrix Applications," *In Proceedings of the Supercomputing Conference*, 1997.

[13] Chun-Yuan Lin, Jen-Shiuh Liu, and Yeh-Ching Chung, "Efficient Representation Scheme for Multi-Dimensional Array Operations, " *IEEE Transactions on Computers*, vol.51, no. 3, pp.327-345, March 2002.

[14] Jen-Shiuh Liu, Jiun-Yuan Lin, and Yeh-Ching Chung, "Efficient Parallel Algorithms for Multi-Dimensional Matrix Operations," *Proceedings of IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, Dec. 2000, Dallas, USA.

[15] Peter D. Sulatycke and Kanad Ghose, "Caching Efficient Multithreaded Fast Multiplication of Sparse Matrices," *In Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, 1998.

[16] Manuel Ujaldon, Emilio L. Zapata, Shamik D. Sharma, and Joel Saltz, "Parallelization Techniques for Sparse Matrix Applications," *Journal of parallel and distribution computing*, 1996.

Table 6: The execution time for the *matrix-matrix addition* operation by compressing one sparse array.

Schemes		*CRS*			*ECRS*	
Sparse Ratios	*Array Sizes*	*C-N IJK*	*C-N IKJ*	*N-N*	*C-N*	*N-N*
0.1	10×10×10	0.027	0.027	0.156	0.025	0.125
	100×100×100	26.599	26.166	160.421	21.797	134.796
	200×200×200	219.133	217.494	1280.544	174.875	1088.741
0.01	10×10×10	0.007	0.007	0.154	0.007	0.123
	100×100×100	4.447	3.907	156.594	3.762	129.826
	200×200×200	33.392	32.886	1245.941	28.305	1044.63
0.001	10×10×10	0.006	0.006	0.154	0.006	0.124
	100×100×100	0.492	0.484	154.613	0.479	130.545
	200×200×200	4.043	3.985	1234.613	3.879	1030.545

Time: *ms*

**C-N*: *Compressed array-Non-compressed array addition*. **N-N*: *Non-compressed array-Non-compressed array addition*.

Table 8: The execution time of the *matrix-matrix addition* operation by compressing two sparse arrays.

Schemes		*CRS*		*ECRS*	*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*C-C IJK*	*C-C IKJ*	*C-C*	*C-C JIK*	*C-C JKI*	*C-C*
0.1	10×10×10	0.096	0.087	0.07	0.135	0.118	0.105
	100×100×100	106.702	91.38	66.03	110.376	106.695	67.540
	200×200×200	873.434	828.625	524.72	897.346	868.475	609.974
0.01	10×10×10	0.017	0.015	0.014	0.023	0.021	0.017
	100×100×100	9.178	8.816	7.838	11.911	11.169	8.445
	200×200×200	78.243	73.842	55.5	80.763	78.935	65.247
0.001	10×10×10	0.009	0.009	0.009	0.013	0.012	0.011
	100×100×100	0.794	0.775	0.718	1.040	0.911	0.837
	200×200×200	6.723	6.655	6.598	7.078	6.918	6.614

Time: *ms*

**C-C*: *Compressed array-Compressed array addition*.

Table 9: The execution time of the *matrix-matrix multiplication* operation by compressing one sparse array.

Schemes		*CRS*		*ECRS*	*CCS*		*ECCS*
Sparse Ratios	*Array Sizes*	*C-N IJK*	*C-N IKJ*	*C-N*	*C-N JIK*	*C-N JKI*	*C-N*
0.1	10×10×10	0.296	0.295	0.276	0.234	0.232	0.224
	100×100×100	3597.635	3335.909	3096.54	2600.032	2578.901	2316.291
	200×200×200	161984.8	150129.8	138897.2	41779.968	41029.251	36921.883
0.01	10×10×10	0.037	0.035	0.035	0.029	0.029	0.029
	100×100×100	455.157	419.873	395.519	265.967	256.444	231.658
	200×200×200	41274.71	40388.59	38301.28	4164.510	4147.689	3689.118
0.001	10×10×10	0.009	0.009	0.009	0.007	0.000007	0.000006
	100×100×100	44.556	44.102	43.102	24.813	24.534	22.422
	200×200×200	729.424	722.765	712.617	407.339	409.073	369.492

Time: *ms*

**C-N*: *Compressed array-Non-compressed array multiplication*.