Research Journal of Applied Sciences, Engineering and Technology 14(8): 310-319, 2017
DOI:10.19026/rjaset.14.4955
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: April 20, 2017 Accepted: May 23, 2017 Published: August 15, 2017

Research Article

A Novel Compressing a Sparse Matrix using Folding Technique

Ashwak Alabaichi, Amjad Hamead Alhusiny and Elham Mohammed Thabit
Computer Department, Sciences Collage, Kerbala University, Karbala, Iraq

Abstract: There are many application problems that emerge in the areas of engineering simulations, scientific computing, information retrieval and economics which use matrixes where non-zero elements are a significant minority with less than 10%. These are universal in many mathematical and scientific applications. These matrixes enable the reduction of storage and computational requirements by storing and carrying out arithmetic with, only the non-zero elements. This is the sparse matrix which must be compressed for these applications. The sparse matrix compression represents non-zero matrix entries. This study presents a novel algorithm for compressing a sparse matrix, which involves three steps. Firstly it involves the division of sparse matrix into sub-matrixes; secondly conducting several transformations; finally coding them. The novel algorithm is called folding. The compressed matrix reduces memory requirement with a good rate compared with the original sparse matrix. *C++* is used in the implementation of this algorithm.

Keywords: Coding, dense matrix, folding, sparse matrix, transformation, unfolding

INTRODUCTION

Sparse matrixes are normally of considerable size with comparatively limited non-zero elements unlike dense matrixes that possess non-zero elements on almost all positions of the matrix (Kourtis, 2010). Sparse matrixes are present in several applications such as structural analysis, computational fluid dynamics, modeling in economics, numerical analysis, numerical optimization, statistical modeling, analysis of power network, electromagnetics, meteorology, medically-related imaging, data mining, finite-element simulations, systems that support decision making in management practice, simulating of circuits, retrieving of information and many other applications (Farzaneh *et al*., 2009). In most applications involving sparse matrixes the size of the matrix is very large. Storing such large matrixes is impossible, even on super computers. Besides, most of the storage and calculations would be wasted on zeros (Stanimirović and Tasić, 2009). The storage and computational requirements of these matrixes can be minimized by different approaches (Stanimirović and Tasić, 2009; Kourtis, 2010; Neelima and Raghavendra, 2012; Farzaneh *et al*., 2009). Compression can be considered as trading data for computation: which leads to a reduction in data and higher computational over head and more space. There have been several significant developments in sparse matrix computations in recent years and some of them have involved the compression of sparse matrix by minimizing its zero elements which are Compressed Row Storage, the Jagged Diagonal Formator Compressed Diagonal Storage format (Kourtis, 2010; Farzaneh *et al*., 2009). As each method depends on the benefits derived from the characteristic of a particular sparse matrix, hence results vary in levels of space efficiency. Sparse matrixes are operated by direct utilization of their storage formats which should offer economy of storage and computational activity. Sparse matrix has the advantage over dense matrix in that it can handle substantial problems that dense matrix is incapable. Sparse matrixes comprise structured and unstructured types. In the case of a structured matrix its non-zero entries have a consistent pattern along a few diagonals. Also, it could comprise the non-zero elements in blocks such as similarly-sized dense sub matrixes, which shape a consistent pattern along a few block diagonals. In comparison, an irregularly structured matrix has irregular entries (Kourtis, 2010). In this study a novel proposal to compress large sparse matrix into a dense matrix is introduced. This proposal can work on structured and unstructured sparse matrixes. Early results indicate that this proposed algorithm possesses excellent results that are related to memory requirement.

LITERATURE REVIEW

The related works are as follows: Firstly: the initial works relate to the improved application performance that is sparse-based on different representations numbering more than thirteen. Compressed Sparse Row

Corresponding Author: Ashwak Alabaichi, Computer Department, Sciences Collage, Kerbala University, Karbala, Iraq

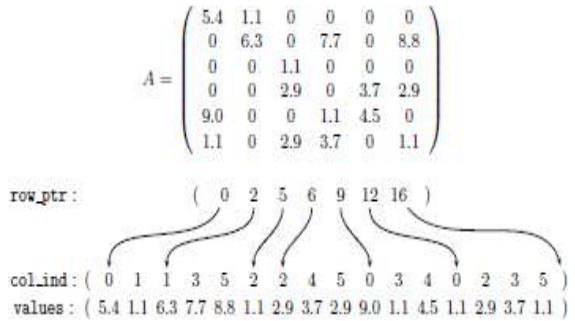

Fig. 1: CSR format

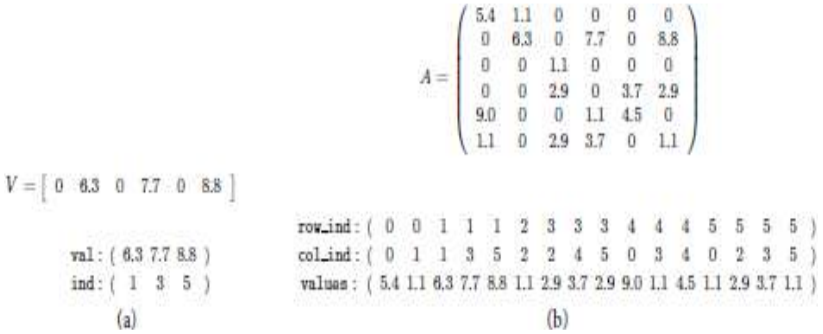

Fig. 2: COO format

(CSR), Compressed Sparse Column (CSC) and Coordinate format (COO) are the significant are popularly used storage techniques (Farzaneh *et al*., 2009).

Compressed Sparse Row (CSR) was a proposal of A. Brameller and D.J. Rose, which has beena widely-used method to store very sizeable sparse matrixes.CSR provides storage for the sparse matrixes arranged in a series of sparse vectors (one for each row) and permits random access to whole rows. In specific terms, the storage of the matrix is in three arrays: *values*, *row_ptr* and *col_ind*. The *values* array provides storage for the non-zero elements of the matrix in row-major order, where as the other two arrays provide storage for indexing information: *row_ptr* containing the location of the first (non-zero) element of each row within the values array and *col_ind* containing the column number for every non-zero element. A sample of the CSR format is shown in Fig. 1 (Farzaneh *et al*., 2009; Kourtis, 2010; Stanimirović and Tasić, 2009).

Compressed Sparse Column (CSC) is comparable to CSR, except for the consecutive storage in columns of the non-zero elements (Stanimirović and Tasić, 2009; González-Domínguez *et al*., 2013).

Coordinate Format (COO) is a very simple sparse storage format. In CCO the compression of a sparse matrix is directly transformed from the dense format which retains the non-zero elements together with their corresponding indices in their matrix location. For example, the COO format for a vector is referred to as a compressed sparse vector or just sparse vector, within which the non-zeroes are maintained contiguously in an array *val* and the indices of these elements are maintained in another array *ind*. This means that *val[i]* maintains the element in position *ind[i]*. An instance of the COO format is shown in Fig. 2 (Vazquez *et al*., 2009; Kourtis, 2010).

Secondly: The related works that try to enhance sparse matrixes in different direction; some of them are mentioned as follows:

A new format of the sparse matrix for representation is produced. This format is subjected to graphics processor architecture and gives 2*x* to 5*x* better performance than Compressed Row Format (CSR) and Coordinate Format (COO). It is also 3*x* to 10*x* better in performance in comparison with CSR vector format. Furthermore it provides 10% to 133% improvement in transferring memory (of only access information of sparse matrix) between CPU and GPU (Neelima and Raghavendra, 2012).

A new coefficient and method for storage coefficient of a large sparse matrix was presented. This method is simple and in expensive. The aim of this technique is to decrease the storage of substantial non-symmetric sparse matrixes. Consequently it is shown that the suggested technique is significantly inexpensive in comparison with other available techniques including Coordinate format, Compressed Sparse Row (CSR) format and Modified Sparse Row (MSR) format (Farzaneh *et al*., 2009).

Comprehensive comparison and evaluation of the storage efficiency for various sparse matrices storage such as CSR, Compressed Sparse Column CSC and

COO are presented. The performance results of matrix-vector multiplication using these storage formats are also presented (Stanimirović and Tasić, 2009).

PROPOSED FOLDING TECHNIQUE

When and where this study was conducted: 2017 - Iraq-Kerbala university.

This proposed Folding Technique consists of three steps:

- Dividing sparse matrix into four quarters equivalent to sub matrices
- Applying transformation based on permutation concept with the sub matrices
- Coding the sub matrices. However, before describing the proposal, the three steps are explained in detail in the following sections.

Dividing sparse matrix into four quarters equivalent to sub matrixes: In this step sparse matrix is divided into four quarters named as A, B, C, D. and all dimensions of quarts are equal. In this step all the following cases are taken into considering:

- If the sparse matrix is square and has odd dimensions; one zero column and row are added. Mathematically this can be described as:
  - Let $M_{m\times m}=(m_{ij})$ be a sparse matrix and $m\ is\ even, n=\frac{m}{2}$, then $M=\begin{pmatrix}A & B\\ C & D\end{pmatrix}$, where $A_{n\times n}=(a_{ij}), B_{n\times n}=(b_{ij}), C_{n\times n}=(c_{ij}), D_{n\times n}=(d_{ij})$.
  - Let $M_{m\times m}=(m_{ij})$ be a sparse matrix and $m\ is\ odd, n=\frac{m+1}{2}$, then $M=\begin{pmatrix}A & B\\ C & D\end{pmatrix}$, where $A_{n\times n}=(a_{ij}), B_{n\times n}=(b_{ij}), C_{n\times n}=(c_{ij}), D_{n\times n}=(d_{ij})$.
- If the sparse matrix is not square, the following action is performed:
  - Suggest sparse matrix has dimensions 30, 50; 50 minus 30 is performed which is equal to 20.
  - Add 20 zero rows to the matrix then divide it to four quarters as described above

Mathematically this can be described as:

$$\begin{aligned}&if\ m>n, then\ \exists k\in N\ such\ that\ m\\ &\qquad=n+k \text{ then } M\\ &\qquad=\begin{pmatrix}A & B\\ C & D\end{pmatrix}, \text{where}\\ &A_{n\times n}=(a_{ij}), B_{n\times n}=(b_{ij}), C_{n\times n}=(c_{ij}), D_{n\times n}\\ &\qquad=(d_{ij}).\\ &if\ n>m, then\ \exists k\in N\ such\ that\ n\\ &\qquad=m+k \text{ then } M\\ &\qquad=\begin{pmatrix}A & B\\ C & D\end{pmatrix}, \text{where } A_{n\times n}\\ &\qquad=(a_{ij}), B_{n\times n}=(b_{ij}), C_{n\times n}\\ &\qquad=(c_{ij}), D_{n\times n}=(d_{ij})\end{aligned}$$

Applying transformations based on the premutation with the sub matrixes: A permutation is the simple exchange in the positions of elements within a message, vector and matrix. Mathematically, a permutation process generates a permutation of the input data, that is, the data is simply rearranged. For example, the group of all permutations of n elements is referred to as the symmetric group S_n and it is not difficult to verify that there are $n!$ Permutations in S_n (Davis, 2003; Mulholland, 2013). In this proposal, permutation is conducted on the elements of four sub-matrixes as transformation. Briefly, we will explain these transformations on one of the sub-matrixes called A.
In mathematical notation, consider:

$$A_{n\times n}=\begin{pmatrix}a_{11} & a_{12}\dots & a_{1n}\\ a_{21} & a_{22}\dots & a_{2n}\\ \vdots & \vdots\ \vdots & \vdots\\ a_{n1} & a_{n2}\dots & a_{nn}\end{pmatrix}$$

Is square matrix and P is resultant matrix after conducting transformations on the elements of A.
Four transformations were conducted as follows:

$$P_1(a_{ij})=a_{ji}, \forall i,j=1,2,\dots,n$$

This transformation replaces all the rows of a given matrix with columns and vice-versa as follows:

$$P_1(A)=\begin{pmatrix}a_{11} & a_{21}\dots & a_{n1}\\ a_{12} & a_{22}\dots & a_{n2}\\ \vdots & \vdots\ \vdots & \vdots\\ a_{1n} & a_{2n}\dots & a_{nn}\end{pmatrix}$$

$$P_2(a_{ij})=a_{n-(i-1)j}, \forall i,j=1,2,\dots,n$$

This transformation reverses the order of the columns. First column will be last column and vice-versa as follows:

$$P_2(A)=\begin{pmatrix}a_{n1} & a_{(n-1)1}\dots & a_{11}\\ a_{n2} & a_{(n-1)2}\dots & a_{12}\\ \vdots & \vdots\ \vdots & \vdots\\ a_{nn} & a_{(n-1)n}\dots & a_{1n}\end{pmatrix}$$

$$P_3(a_{ij})=a_{in-(j-1)}, \forall i,j=1,2,\dots,n$$

This transformation reverses the order of the rows. First row will be last row and vice-versa as follows:

$$P_3(A)=\begin{pmatrix}a_{1n} & a_{2n}\ \dots & a_{nn}\\ a_{1(n-1)} & a_{2(n-1)}\dots & a_{n(n-1)}\\ \vdots & \vdots\ \vdots & \vdots\\ a_{11} & a_{21}\dots & a_{n1}\end{pmatrix}$$

$$P_4(a_{ij})=a_{n-(i-1)n-(j-1)}, \forall i,j=1,2,\dots,n$$

This transformation reverses the order of the rows and columns. First row will be last row and vice-versa

as well as the first column will be last column as follows:

$$P_4(A) = \begin{pmatrix} a_{nn} & a_{(n-1)n} & \cdots & a_{1n} \\ a_{n(n-1)} & a_{(n-1)(n-1)} \cdots & a_{1(n-1)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{(n-1)1} \cdots & a_{11} \end{pmatrix}$$

Coding the sub matrixes: Coding involves the organization and categorization of data. Codes offer an option for labeling, compiling and organizing data, which can be carried out in several ways but normally involves categorizing words, phrases, numbered or symbols in various appropriate coding categories systematically (Baralt, 2012). In this proposal coding is defined as follows:

Give code value to every three symmetric elements: $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ are zeroes and one of $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ is not zero for $i, j = 1, 2, \ldots\ldots, n$.

An example of how coding is conducted is shown below:

$0, 0, 0, 1$ give code value equal to 1.
$0, 0, 1, 0$ give code value equal to 2.
$0, 1, 0, 0$ give code value equal to 3.
$1, 0, 0, 0$ give code value equal to 4.
$0, 0, 0, 2$ give code value equal to 5.
$0, 0, 2, 0$ give code value equal to 6 and so on.

The following procedure illustrates the coding operation.

Procedure coding elements of matrixes
Input: a, b, c and d as integer elements
Output: x as integer value
if a, b, c and d are equal 0 then set x = 0
If a, b and c are equal 0 and d greater than 0 then set x = 4*(d-1) +1

If a, b and d are equal 0 and c greater than 0 then set x = 4*(c-1) +2

If a, c and d are equal 0 and b greater than 0 then set x = 4*(b-1) +3

If b, c and d are equal 0 and a greater than 0 then set x = 4*(a-1) +4

Describing the folding technique: In this technique the sparse matrix is divided into four sub-matrixes as described in above Section. Coding procedure will be conducted on the elements of the sub-matrixes, of which at least every three elements must equal to zero as described in above Section, otherwise use transformations to change positions of the elements in each sub-matrix as described in above section. This transformation which satisfies the case that at least three of the symmetric elements $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ are zeroes for $i, j = 1, 2, \ldots\ldots, n$. Table 1 describes a part from the transformations which is applied on four sub-matrixes. Then coding procedure will be conducted to get new coded matrix. The symbol "---" in Table 1 means other transformations exist between row 19 and row 51, as well as between row 57 and row 81. Folding Algorithm is described in algorithm 1. Unfolding algorithm retrieves the original sparse matrix from compressed matrix as described in algorithm 2.

Table 1: Part of the transformations on the sub-matrixes

No.	Transformations
1	$a_{ij}, b_{ij}, c_{ij}, d_{ij}$
2	$a_{ij}, b_{in-(j-1)}, c_{n-(i-1)\,n-(j-1)}, d_{n-(i-1)\,j}$
3	$a_{ij}, b_{in-(j-1)}, c_{i\,n-(j-1)}, d_{ij}$
4	$a_{ij}, b_{n-(i-1)j}, c_{ij}, d_{n-(i-1)j}$
5	$a_{n-(i-1)n-(j-1)}, b_{ij}, c_{i\,j}, d_{i\,j}$
6	$a_{ij}, b_{n-(i-1)n-(j-1)}, c_{i\,j}, d_{ij}$
7	$a_{ij}, b_{ij}, c_{n-(i-1)n-(j-1)}, d_{i\,j}$
8	$a_{ij}, b_{ij}, c_{ij}, d_{n-(i-1)n-(j-1)}$
14	$a_{ij}, b_{ij}, c_{n-(i-1)n-(j-1)}, d_{n-(i-1)n-(j-1)}$
15	$a_{ij}, b_{n-(i-1)n-(j-1)}, c_{n-(i-1)n-(j-1)}, d_{n-(i-1)n-(j-1)}$
16	$a_{n-(i-1)n-(j-1)}, b_{ij}, c_{n-(i-1)n-(j-1)}, d_{n-(i-1)n-(j-1)}$
17	$a_{n-(i-1)n-(j-1)}, b_{n-(i-1)n-(j-1)}, c_{i\,j}, d_{n-(i-1)n-(j-1)}$
18	$a_{n-(i-1)n-(j-1)}, b_{n-(i-1)n-(j-1)}, c_{n-(i-1)n-(j-1)}, d_{ij}$
19	$a_{ij}, b_{n-(i-1)n-(j-1)}, c_{n-(i-1)n-(j-1)}, d_{n-(i-1)n-(j-1)}$
---	---
---	---
51	$a_{ji}, b_{ji}, c_{ij}, d_{ij}$
52	$a_{ji}, b_{ij}, c_{ji}, d_{ij}$
53	$a_{ji}, b_{ij}, c_{ij}, d_{ji}$
54	$a_{ij}, b_{ji}, c_{ji}, d_{ij}$
55	$a_{ij}, b_{ji}, c_{ij}, d_{ji}$
56	$a_{ij}, b_{ij}, c_{ji}, d_{ji}$
57	$a_{ji}, b_{ji}, c_{ji}, d_{ij}$
---	---
---	---
81	$a_{n-(i-1)n-(j-1)}, b_{ij}, c_{ij}, d_{ji}$
82	$a_{ji}, b_{n-(i-1)n-(j-1)}, c_{ij}, d_{ij}$
83	$a_{ij}, b_{n-(i-1)n-(j-1)}, c_{ji}, d_{ij}$
84	$a_{ij}, b_{n-(i-1)n-(j-1)}, c_{ij}, d_{ji}$
85	$a_{ji}, b_{ij}, c_{n-(i-1)n-(j-1)}, d_{ij}$
86	$a_{ij}, b_{ji}, c_{n-(i-1)n-(j-1)}, d_{ij}$
87	$a_{ij}, b_{ij}, c_{n-(i-1)n-(j-1)}, d_{ji}$

Folding algorithm 1:

1. Input: X is a sparse array.
2. Output: Y is coded array of X, T is a list of transformer numbers and D1 is a list of dimensions of array of each division.
3. Divide array X into 4 quarters, each quarter is a square array and all of them have same size as described in Section above.
4. Store first quarter in array A, second quarter in array B, third quarter in array C and last quarter in array D.
5. Find the transformation that can be applied to all elements in arrays A, B, C and D.
6. If the transformation exists, encode the elements in arrays A, B, C, D using encoding procedure, set X=Y, store the transformation number in list T, store the dimension of Y in list D1.
7. If the transformation does not exist, end algorithm.
8. Go to 3.

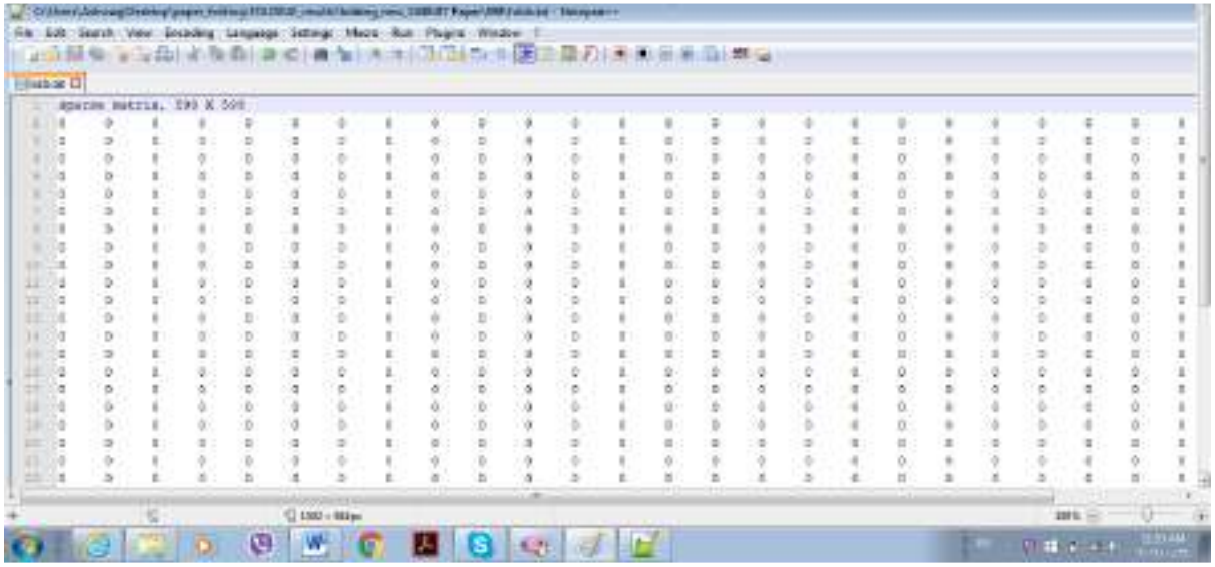

Fig. 3: Sparse matrix with 599x599

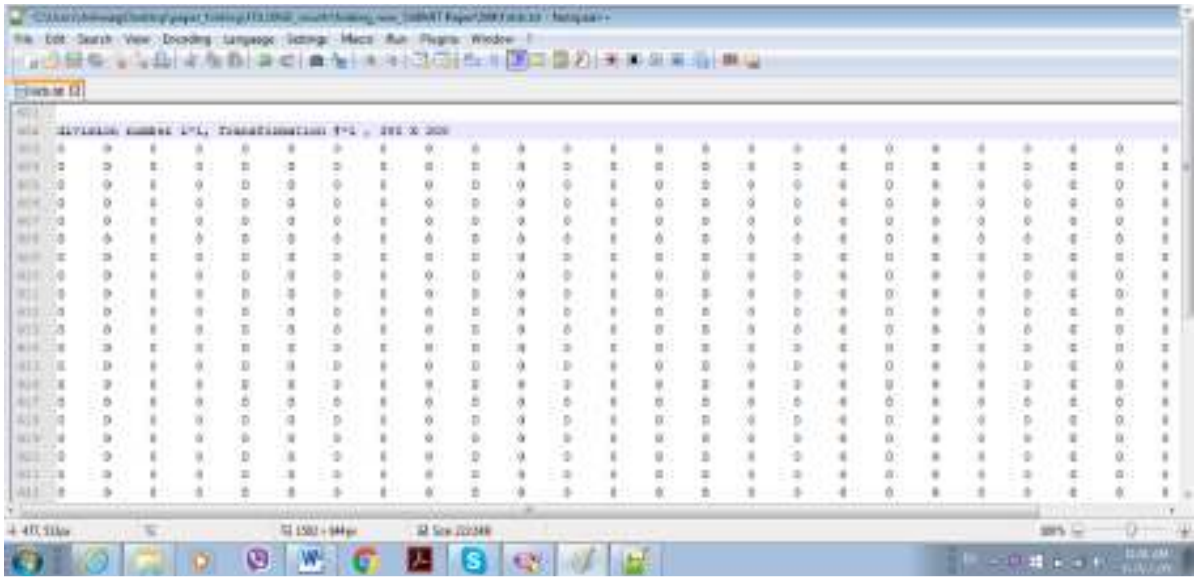

Fig. 4: Sparse matrix with 300x300 in first division

Unfolding algorithm 2:

1. Input: Y is coded array and T is a list of the transformation numbers of each division and D1 is a list of dimensions of array of each division.
2. Output: X is decoded array of Y.
3. For each transformation number in list T after applying it, perform steps 4 to 6.
4. Find Arrays A, B, C and D from array Y using decoding procedure.
5. Merge the arrays A, B, C and D to get array X.
6. Set Y = X.

Procedure in decoding elements of matrixes:

Input: x is integer number
Output: A, B, C and D are integer numbers
Set a = 0, b = 0, c = 0 and d = 0
If x> 0 and x mod 4 = 0 then set A= x/4
If x> 0 and x mod 4 = 1 then set B = x/4 +1
If x> 0 and x mod 4 = 2 then set C = x/4 + 1
If x> 0 and x mod 4 = 3 then set D = x/4 +1
End.

EXPERIENTIAL RESULTS

In this section several experiments are carried out to compress sparse matrix with different dimensions as follows:

Figure 3 shows part of sparse matrix with dimension 599x599, where as Fig. 4 to 7 show the first to fourth divisions with dimensions 300x300, 150x150, 75x75 and 38x38 respectively. In Fig. 8 the dimensions

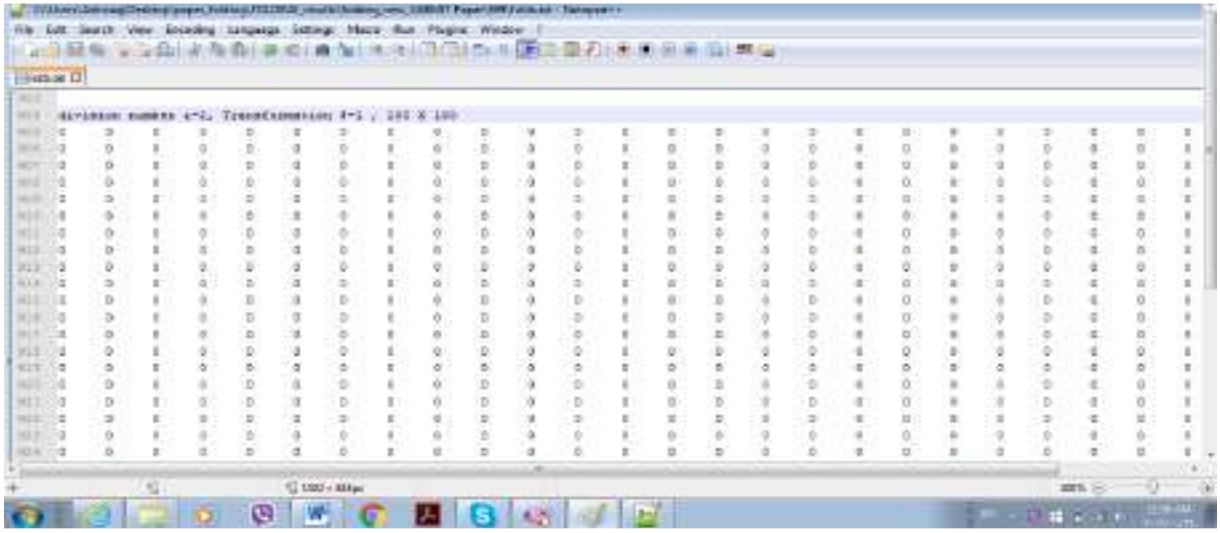

Fig. 5: Sparse matrix with 150x150 in second division

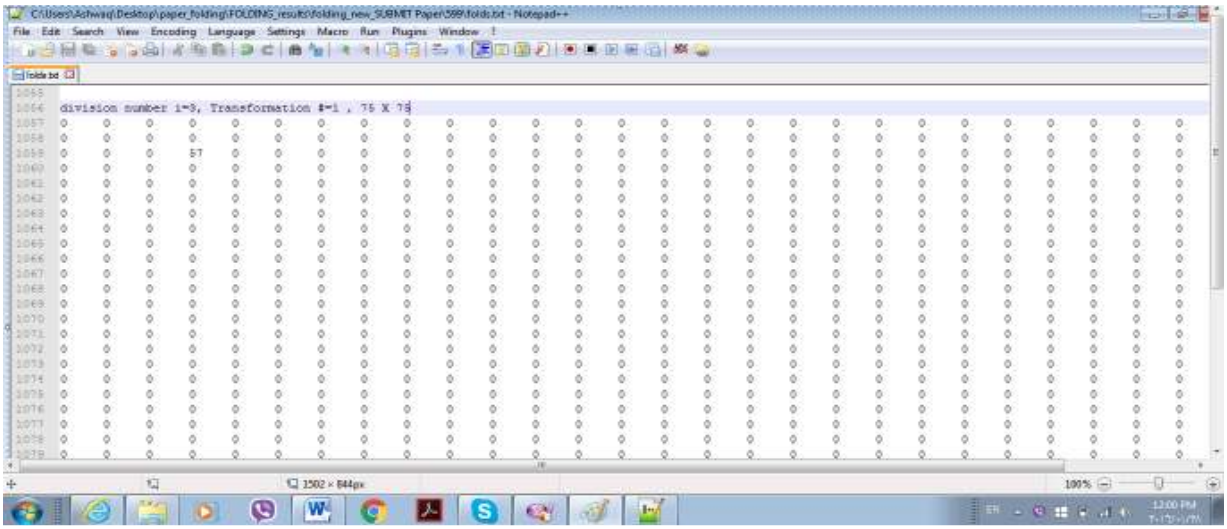

Fig. 6: Sparse matrix with 75x75 in third division

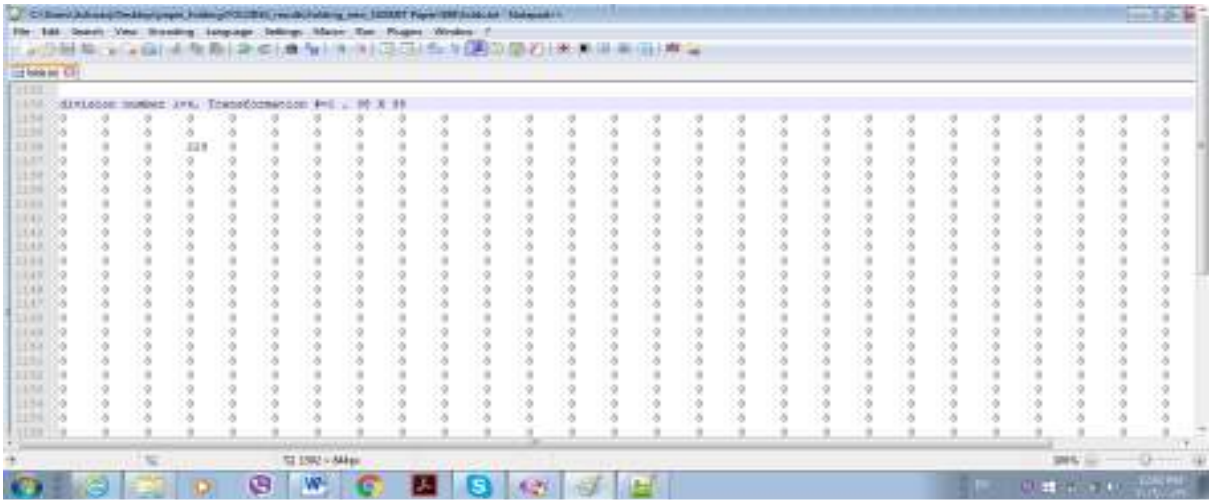

Fig. 7: Sparse matrix with 38x38 in fourth division

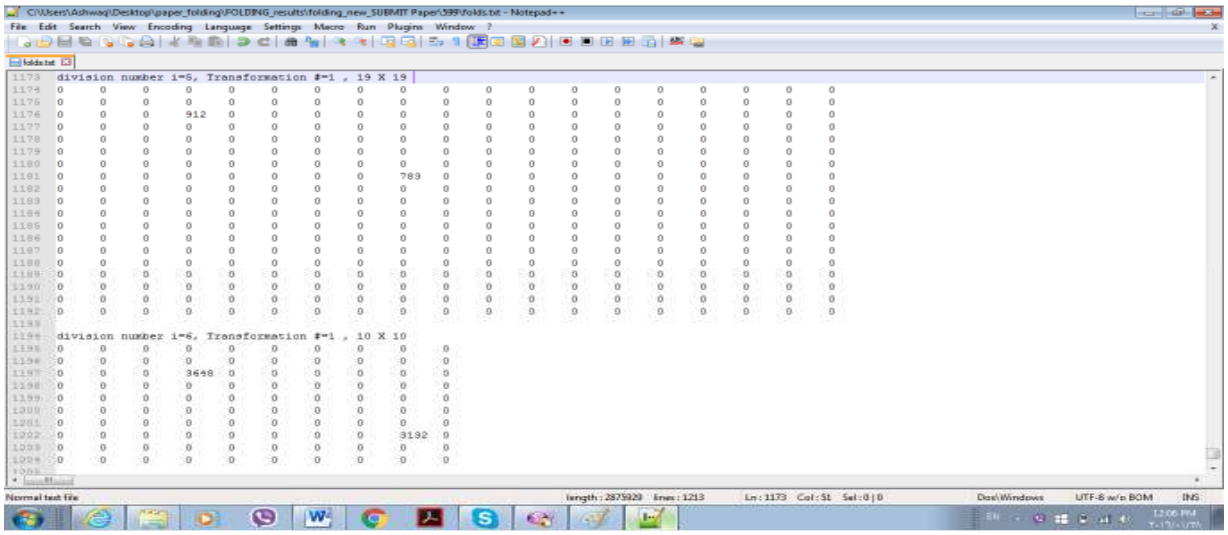

Fig. 8: Divison stages 5 and 6

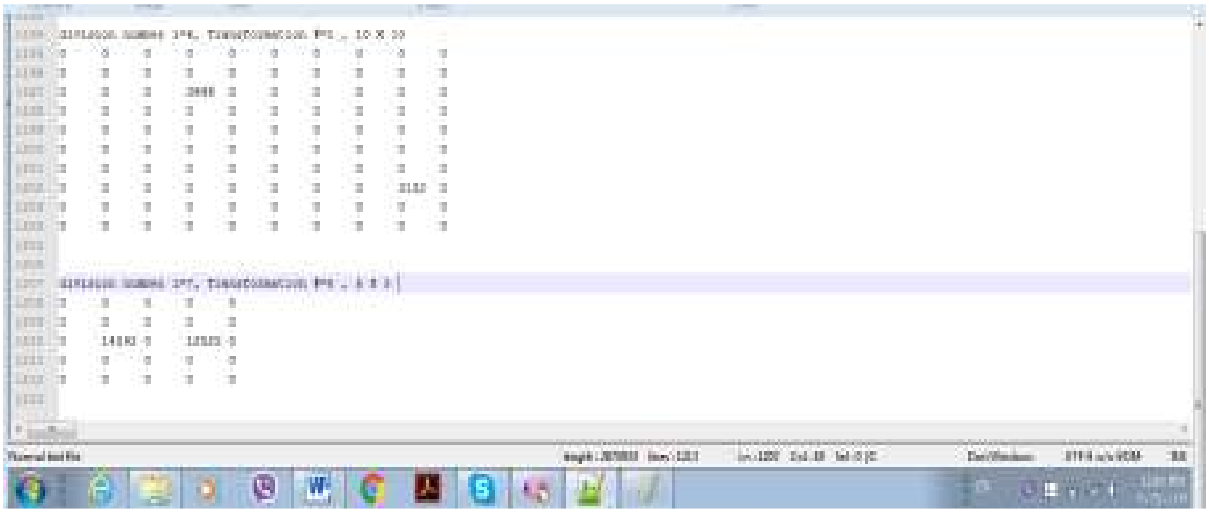

Fig. 9: Divison stages 6 and 7

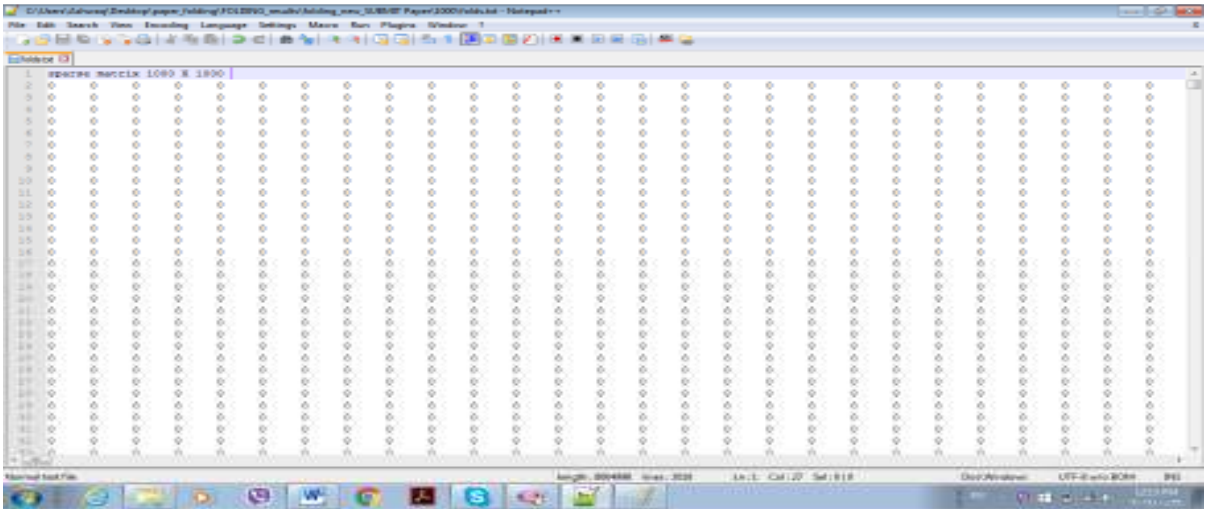

Fig. 10: Sparse matrix with 1000x1000

of the fifth and sixth divisions were 19x19 and 10x10 respectively. Figure 9 shows the sixth and seventh divisions with dimensions 10x10 and 5x5 respectively. The number of transformations applied was 1 in all divisional stages except the seventh division which was 5. The compress matrix was the matrix in the seventh division with dimension 5x5.

Figure 10 shows part of sparse matrix with dimension 1000x1000, while Fig. 11 to 15 shows the first to fifth divisions with dimensions 500x500, 250x 250, 125x125, 63x63, 32x32 respectively. Figure 16 illustrates the sixth and seventh divisions which have dimensions 16x16 and 8x8 respectively. The number

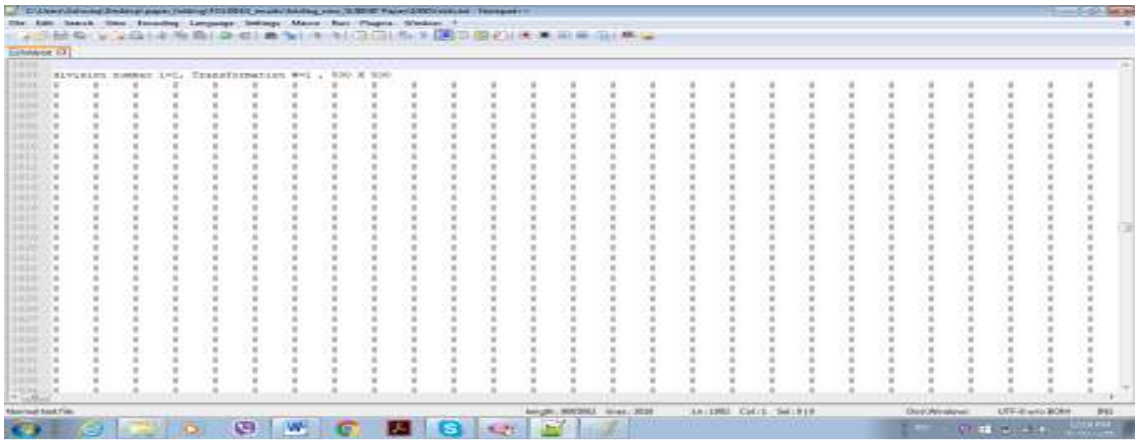

Fig. 11: Sparse matrix with 500x500 in first division

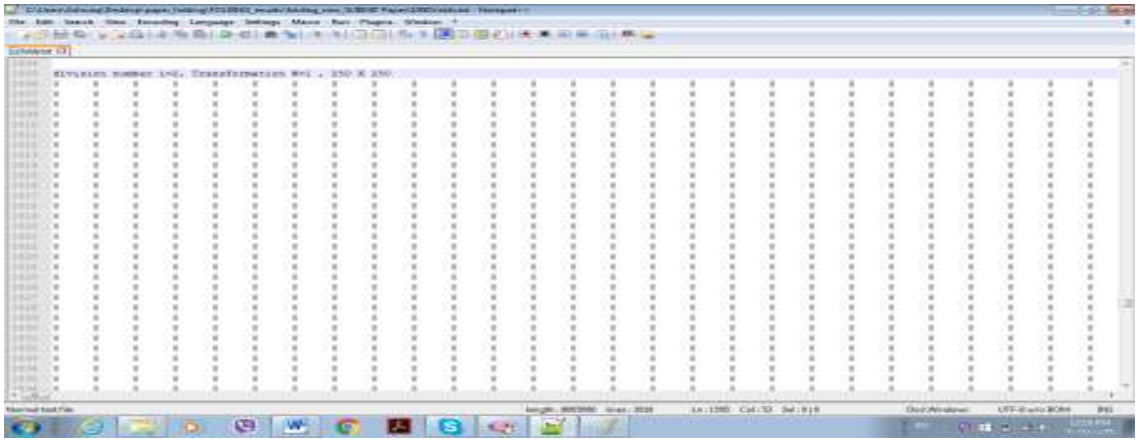

Fig. 12: Sparse matrix with 250x250 in second division

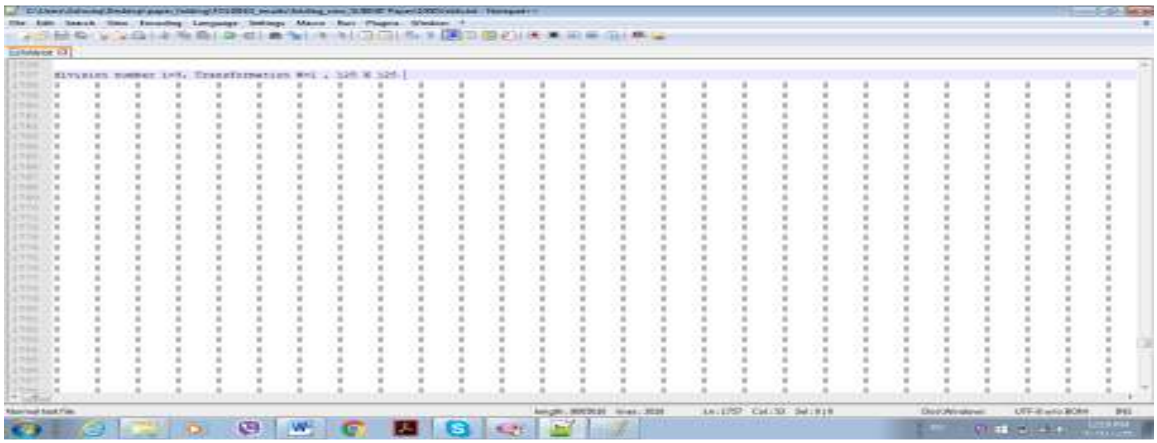

Fig. 13: Sparse matrix with 125x125 in third division

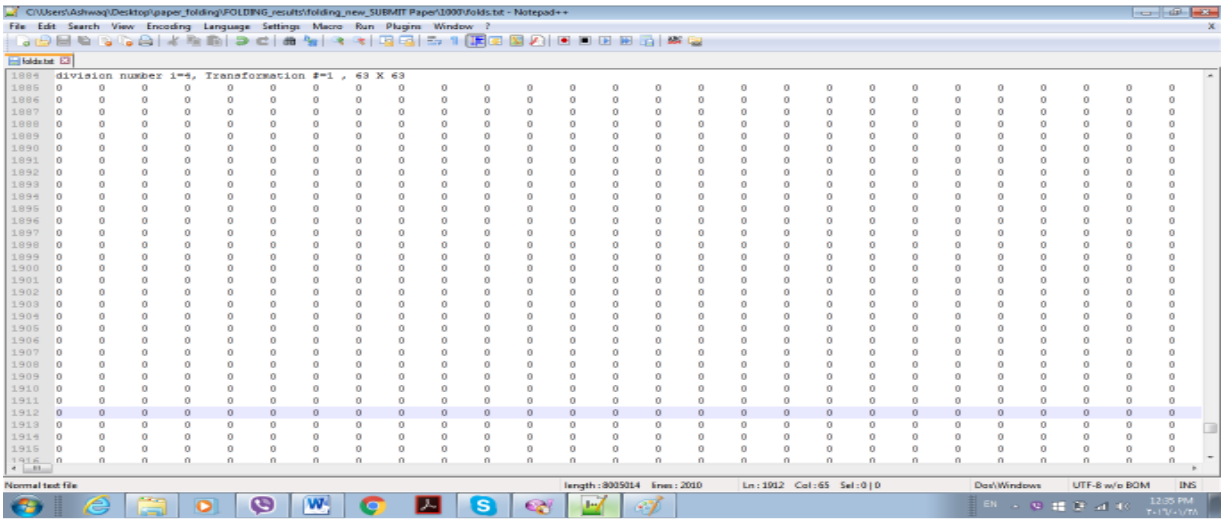

Fig. 14: Sparse matrix with 63x63 in fourth division

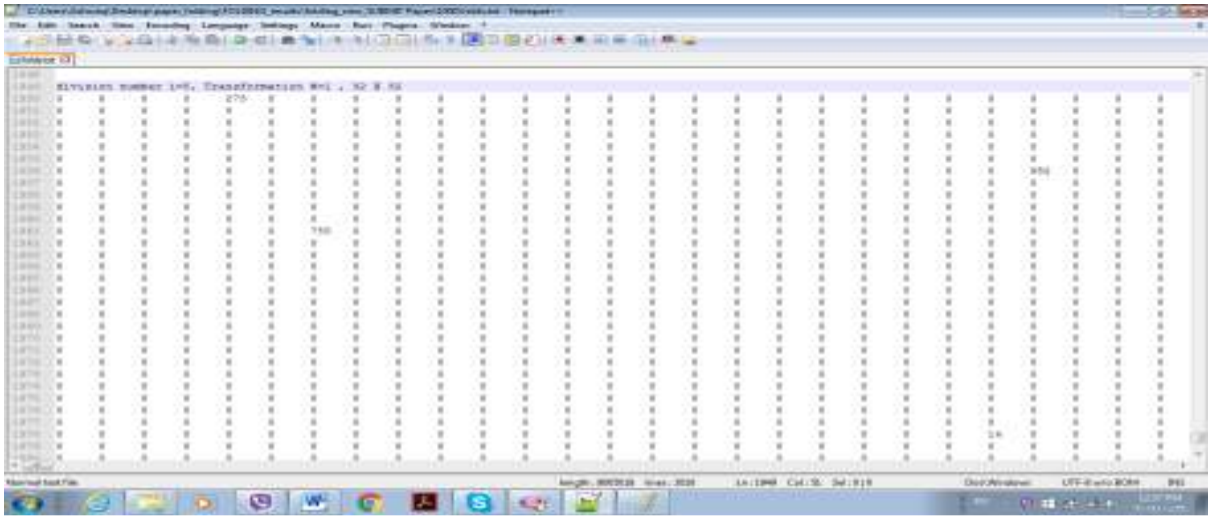

Fig. 15: Sparse matrix with 32x32 in fifth division

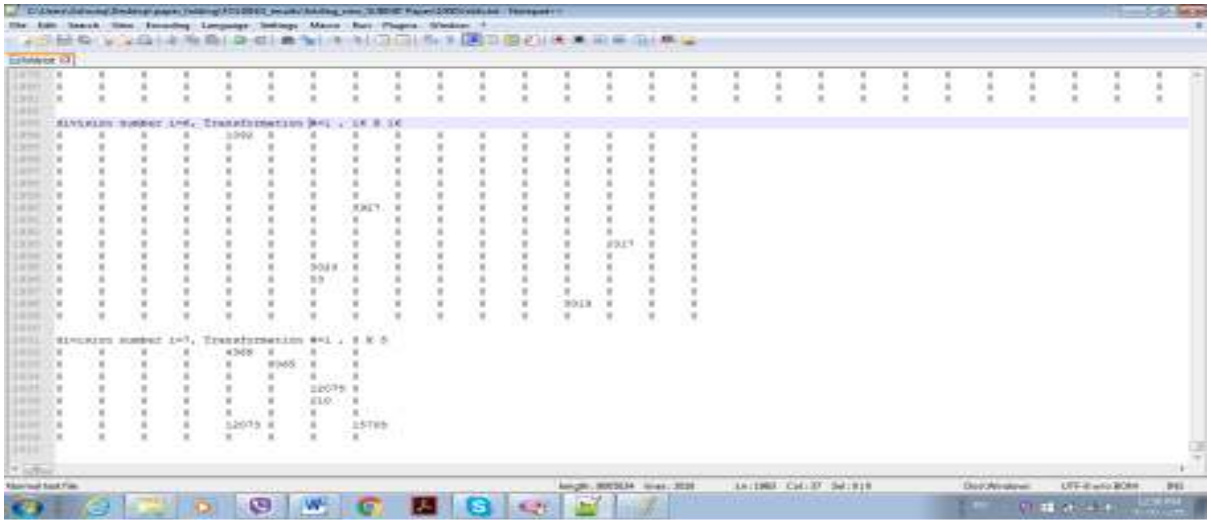

Fig. 16: Divison stages 6 and 7 with compressed matrix

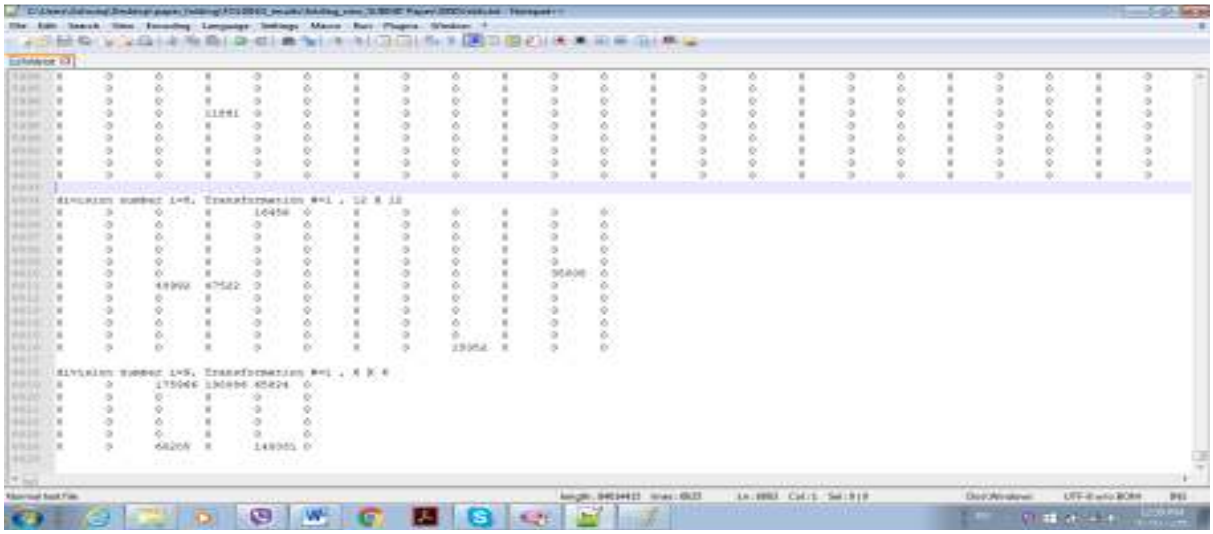

Fig. 17: Eighth and ninth divisions

of the transformation applied was 1 in all divisional stages. The compress matrix was the matrix in the seventh division with dimension of 8x8.

Briefly the sparse matrix with dimension 3000 x 3000 is present only in the two last divisions where the compressed matrix resulted from the ninth division with dimension of 6x6 as shown in Fig. 17.

From the above results it can be noticed that any matrix can be compressed with any dimension.

CONCLUSION AND RECOMMENDATIONS

The research addressed the limitation of the sparse matrix which utilizes a large memory to store numerous zero elements. It is unsuitable for small devices with limited memory. The novel algorithm should satisfy the important requirement which is reducing memory requirement. The sparse matrix requires memory 4.76 MB with dimension 1000 x 1000 while after compressing it requires memory 400 bytes only. Also the sparse matrix requires 6 MB with dimension 3000 x 3000 but after compressing it requires 264 bytes only. It can be shown the memory requirements decreased when the size of the sparse matrix is compressed.

Future research could investigate the reduction of memory requirements and overhead in computation by compressing the sparse matrix through one division only.

REFERENCES

Baralt, M., 2012. Coding Qualitative Data. In: Mackey, A. and S.M. Gass (Eds.), Research Methods in Second Language Acquisition. Wiley-Blackwell, Malden, pp: 222-244.

Davis, T., 2003. Permutation Groups. No. 3, pp: 1-12. Retrieved from: http://www.geometer.org/mathcircles/perm.pdf.

Farzaneh, A., H. Kheiri and M.A. Shahmersi, 2009. An efficient storage format for large sparse matrices. Commun. Fac. Sci. Univ., Ank. Series A1, 58(2): 1-10.

González-Domínguez, J., Ó. García-López, G.L. Taboada, M.J. Martín and J. Touriño, 2013. Performance evaluation of sparse matrix products in UPC. J. Supercomput., 64(1): 100-109.

Kourtis, A.K., 2010. Data compression techniques for performance improvement of memory-intensive applications on shared memory architectures. Ph.D. Thesis, Athens, pp: 1-109. Retrieved from: http://www.cslab.ntua.gr/~kkourt/phd/phd-en.pdf.

Mulholland, J., 2013. Permutation puzzles: A mathematical perspective. Department of Mathematics, Simon Fraser University, pp: 1-312.

Neelima, B. and P.S. Raghavendra, 2012. Effective sparse matrix representation for the GPU architectures. Int. J. Comput. Sci. Eng. Appl., 2(2): 151-165.

Stanimirović, I.P. and M.B. Tasić, 2009. Performance comparison of storage formats for sparse matrices. Facta Univ., (NIˇS) Ser. Math. Inform., 24: 39-51.

Vazquez, F., E.M. Garzon, J.A. Martinez and J.J. Fernandez, 2009. The sparse matrix vector product on GPUs. Proceeding of the International Conference on Computational and Mathematical Methods in Science and Engineering, pp: 1-13.