SPARSE MATRIX STORAGE USING DECIMAL CODING

V. Kabeer[1], Afsal K[2], Sainul Abideen N[3]

[1]Farook College, Kozhikode
[2]Farook College, Kozhikode
[3]Farook College, Kozhikode

Abstract

Sparse Matrices are having very important role in computation, especially managing linear algebraic systems. Its used from storage to computation for different applications like solving partial differential equations, computer graphics, Machine learning, maps and graphs etc. Heavy usage of sparse matrix for different computational applications in a wide variety of areas make optimization of its storage very important. There are several models developed for representing sparse matrices in memory efficiently and many researchers are working on it. here, We are proposing a new method for storing sparse matrices efficiently by using very less storage space.

Keywords:- *Sparse Matrix Storage, Sparse Matrix, Decimal Coding, Matrix Storage Using Decimal Coding, Dense Matrix Storage.*

---***---

1. INTRODUCTION

Sparse matrices have huge applications from computer processing[1][2][3][6] to data storage[10]. Sparse matrices are matrices with a considerable number of zero elements. As the matrices having very less number of nonzero elements, scientists developed several methods for efficient storage of the matrix in computer memory by eliminating zero from storage[10][8]. Basic idea is to store nonzero elements with its positions in the matrix so that we can easily reproduce the matrix by using the stored information. There are several methods developed for storing sparse matrix such as Dictionary of Keys (DOK), Yale sparse matrix format, Compressed Raw Storage (CRS), Compressed Sparse Column(CSC), etc.

Here, we are proposing a new method for sparse matrix storage which found more efficient in memory utilization. Most of the test cases shows considerably less memory space to store a sparse matrix by using this method instead of using conventional methods. Moreover sparse matrices, we tried the method with a slight modification for normal dense matrix which is having any particular non zero element repeating instead of zero, and we were able to store that matrix too with considerably reduced storage space.

Basic idea of sparse matrix storage is to store only nonzero elements in memory with its address (positions) and avoid zero elements to reduce memory space[5]. So, if N is the number of nonzero elements in a sparse matrix, V will be an array of nonzero elements and R will be the row reference array which will be stored the raw reference and C will be the column reference array which will be containing column reference. So in the normal method, if N be the number of nonzero elements, 3N will be the storage space needed to store the matrix. By getting the nonzero values and references, we can regenerate the sparse matrix anytime in need. We are able to do basic matrix operations also by using the values stored in memory.

2. PRIOR ART

2.1 Sparse Matrix - Conventional Model

A matrix is typically stored as a two-dimensional array. Each entry in the array represents an element a (i,j) of the matrix and is accessed by the two indices i and j. Conventionally, i is the row index, numbered from top to bottom, and j is the column index, numbered from left to right. For an m X n matrix, the amount of memory required to store the matrix in this format is proportional to m X n (disregarding the fact that the dimensions of the matrix also need to be stored).

In the case of a sparse matrix, substantial memory requirement reductions can be realized by storing only the non-zero entries. Depending on the number and distribution of the non-zero entries, different data structures can be used and yield huge savings in memory when compared to the basic approach. The trade-off is that accessing the individual elements becomes more complex and additional structures are needed to be able to recover the original matrix unambiguously. Even then,its widely used to reduce memory space to store the matrix for different computational purpose.

2.1 CSR, CRS and Yale Format

The Compressed Sparse Row (CSR) or Compressed Row Storage (CRS) format represents a matrix M by three (onedimensional) arrays, that respectively contain nonzero values, the extents of rows, and column indices [10][11]. It is almost similar to COO method, but compresses the row indices, hence the name. This format allows fast row access and matrix-vector multiplications. The CSR format has been in use since at least the mid-1960s, with the first complete description appearing in 1967. This scheme is preferred over the coordinate scheme because it is often more useful for performing typical computations[11]. The CSR format stores a sparse m X n matrix M in row form using three (onedimensional) arrays (A, IA, JA). It stores the subsequent nonzeros of the matrix rows in contiguous memory locations. Assuming we have a non symmetric sparse matrix A, we create three vectors : one for floating point numbers (val) and the other two for integers. The val vector stores the values of the nonzero elements of the matrix A as they are traversed in a row-wise fashion. The col vector stores the column indexes of the elements in the val vector. The storage savings for this approach is significant. Instead of storing n pow(2) elements, we need only 2nnz+n+1 storage locations.

3. METHOD

Consider a mXn sparse matrix A

$$A = \begin{bmatrix} a_{11} & a_{12} & . & . & a_{1n} \\ a_{21} & a_{22} & . & . & a_{2n} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{m1} & a_{m2} & . & . & a_{mn} \end{bmatrix}$$

where m is the number of rows and n is the number of columns and nnz is the number of non zero elements in the matrix A,

the sparse matrix A can be represented as two one dimensional arrays P, V as follows

where array P stores the horizontal / vertical weight of each row / column. The first position of P will be stored with the number of row / column of the given matrix A

P = [m/n, w1, w2, w3, ... wn]

and array V stores all non zero elements in the given matrix A in row/column major order.

V = [nz1, nz2, nz3, ... nzn]

Weight Calculation

Consider row major weight calculation :- Let n is 4, we take the first row r1= [a11 a12 a13 a14], where a11, a12, a14 are zeros and a13 is a non zero element, then we consider the nonzero element as 1 so the row r1 will be [0 0 1 0], we consider it as a binary number and convert it to decimal equivalent. Hence the weight of the row will be 2. For calculating weight in column major, we have to do the same procedure with each column in the matrix. we can save space by selecting smallest from no of rows and no of columns.

4. EXAMPLE

We can check the new method of storage by converting few matrices to the new format. Here, we are considering a normal 3x3 sparse matrix with four nonzero elements, a 3x3 Dense matrix with more repeated elements, and a 3x10 sparse matrix. We are introducing an intermediate step Bi for easy understanding of the process (It can be avoided in actual implementation).

4.1 Normal Sparse Matrix

$$A = \begin{bmatrix} 32 & 0 & 0 \\ 0 & 86 & 8 \\ 0 & 26 & 0 \end{bmatrix}$$

We can construct Binary matrix of A by considering all non zero values as 1 and zero values as zero itself.

$$Bi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Now find decimal value of each row of binary matrix.

$$P = \begin{bmatrix} 4 \\ 3 \\ 2 \end{bmatrix}$$

P=

3	4	3	2

V=

32	86	8	26

4.2 Dense Matrix

Generally, a dense matrix is a matrix with o(n2) non zero elements. If the dense matrix is having a considerable number of unique elements or considerable number of repetition, we can convert it in to a sparse kind of format and store it in the method. It will reduce a considerable amount of storage space as it can be stored as a sparse matrix. Normally, its not feasible to store a dense matrix as sparse matrix, but by using this method, its possible by some easy transformation.

$$A = \begin{bmatrix} 22 & 86 & 22 \\ 98 & 22 & 22 \\ 22 & 22 & 54 \end{bmatrix}$$

We can construct Binary matrix of A by considering all non zero values as 1 and zero values as zero itself.

$$B = \begin{bmatrix} 0 & 86 & 0 \\ 98 & 0 & 0 \\ 0 & 0 & 54 \end{bmatrix}$$

Now find decimal value of each row of binary matrix.

$$B = \begin{bmatrix} 2 \\ 4 \\ 1 \end{bmatrix}$$

P=

3	22	2	4	1

V=

86	98	54

4.3 3x10 Sparse Matrix

If we consider a 3X10 matrix, we can understand the ability of our new storage method to store the sparse matrix with very limited number of storage space. As we have option to select row major/ column major weight calculation method, we can use row major method here, as its only having 3 rows. Beauty of this scenario is that we can store the position of entire 3X10 matrix by using only 3 storage space. Total space need to store nonzero elements is nnz. so total space needed to store the 3X10 sparse matrix is 3+nnz, which is very much storage space efficient compared to normal method and Yale method.

$$A = \begin{bmatrix} 56 & 0 & 0 & 0 & 0 & 91 & 0 & 0 & 0 & 0 \\ 0 & 0 & 74 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 66 & 0 & 0 & 0 & 0 & 99 \end{bmatrix}$$

We can construct Binary matrix of A by considering all non zero values as one and zero values as zero itself.

$$Bi = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now find decimal value of each row of binary matrix.

$$B = \begin{bmatrix} 528 \\ 128 \\ 33 \end{bmatrix}$$

P=

3	528	128	33

V=

56	91	74	66	99

5. COMPARISON

When we compare the new method with the conventional methods, it provide more space optimized storage of sparse matrix. Consider a m X n sparse matrix A, where m is the number of rows and n is the number of columns, with nnz non zero elements. Total space used to store the sparse matrix will be 3 X nnz in normal method, where each space will be used to store non zero element, row position and column position of the element.

In our method, position of all non zero elements in a row/column will be stored in a single number. So, total space needed to store address of a complete sparse matrix is no of rows/no of columns (whichever is lesser). So in the above example, total space needed to store the matrix is small(no of rows/ no of columns) + nnz + 1, which will be smaller than conventional method in most cases.

6. OPERATIONS

After converting a sparse matrix to this format, its possible to perform basic operations without converting back to the original form and it saves computation power and time considerably. We can perform most of the basic operations like Addition, Subtraction, Transpose, etc.

6.1 Addition

We can perform addition of two sparse matrices stored in this method without converting it to actual format. Here we assume both matrices are stored in the same format and both are used either row major or column major format for conversion. Positions of nonzero values can be obtained from P by dividing its vale with power of the position and possible to create resultant matrix C which represent as V and P.

6.2 Subtraction

We can perform subtraction of two sparse matrices stored in this method. here too, we assume both matrices are stored in the same format and both are used either row major or column major format for conversion. Subtraction also can be done with the similar procedure we used in addition.

7. THREE DIMENSIONAL MATRIX

The method we used here to store two dimensional sparse matrix can be used to store three dimensional sparse matrices too with some slight changes. in that case P will be a two dimensional matrix with weights of all single rows/ columns and V will be a one dimensional array as in the above case. Reducing a three dimensional matrix in to two dimensional matrix with least order will considerable reduce size of the storage space needed.. We can even extend the method in to n dimensional matrices with slight modifications in the model.

8. CONCLUSION

There are many proven methods developed for sparse matrix storage and already in use for different kind of applications. Each method is having its own benefits and drawbacks but selection of methods for appropriate applications make it perfect. Few applications will be critical in speed but storage space will not be a problem, but other application will be having limited storage resources. Our method is more suitable where storage space is limited. Bringing the method to heavy size matrices and multidimensional matrices will considerably reduce storage space.

REFERENCES

[1]. Sparse Matrix Storage Format, Fethulah Smailbegovic, Georgi N. Gaydadjiev, Stamatis Vassiliadis Computer Engineering Laboratory, Electrical Engineering Mathematics and Computer Science Mekelweg 4, 2628CD Delft TU Delft fethulah@computer.org stamatis, georgi@ce.et.tudelft.nl

[2]. Matrix computing coprocessor for an embedded system.Bin Zhang , Kuizhi Mei, Jizhong Zhao Xian Jiaotong University, Xian 710049, China.

[3]. A hardwaresoftware co-design approach for implementing sparse matrix vector multiplication on FPGAs Shweta Jain-Mendon , Ron Sass Reconfigurable Computing Systems Lab, Department of Electrical Computer Engineering, University of North Carolina at Charlotte, United States.

[4]. Optimization of sparse matrix-vector multiplication using reordering techniques on GPUs Volume 36, Issue 2, March 2012, Pages 65- 77. Juan C. Pichel, Francisco F. Rivera, Marcos Fernndez, Aurelio Rodrguez.

[5]. Information storage and effective data retrieval in sparse matrices. Hans J. Bentz, Michael Hagstroem University of Osnabrueck Germany. Guenther Palm University of Duesseldorf Germany.

[6]. A geometric framework for sparse matrix problems. Gunnar Carlsson, Vin de Silva , silva@math.stanford.edu Department of Mathematics, Stanford, CA, USA. Advances in Applied Mathematics Volume 33, Issue 1, July 2004, Pages 1-25

[7]. Sparse Matrix Technology. von: Sergio Pissanetzky Elsevier Reference Monographs, 1984 ISBN: 9781483270401 , 336 Seiten.

[8]. An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication Autoria: Liu, Weifeng; Vinter, Brian;

[9]. Efficient sparse matrix vector multiplication using cache oblivious extension quadtree storage format. Future Generation Computer Systems. Jilin Zhang , Jian Wan, Fangfang Li , Jie Mao d, Li Zhuang , Junfeng Yuan , Enyi Liua,, Zhuoer Yua.

[10]. Sparse Matrices and their Applications. ISBN13 9781461586777, Publisher :-Springer-Verlag New York Inc.

[11] Iterative Methods for Sparse Linear Systems. by Yousef Saad, Second edition with corrections. January 3rd, 2000.