|  |  |
| --- | --- |
|  |  |

The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program   
of the Lobachevsky State University of Nizhni Novgorod   
among the world's research and education centers

Strategic initiative  
“Achieving leading positions in the field of supercomputer technology   
and high-performance computing”

**DIRECT METHODS   
FOR SOLVING SYSTEM OF LINEAR EQUATIONS**

*Lecture 5. Sparse matrix storage formats.   
Implementation of standard matrix operations.*

Nizhni Novgorod

2014

# Objectives

The objective of the lecture is the investigation of the sparse matrix storage formats and also the algorithms implementing standard matrix operations. The efficiency of the proposed algorithms that take into account matrix sparseness is emphasized.

# Abstract

The storage formats and the standard operations for the sparse matrices are investigated. The notion of a sparse matrix is introduced, the fundamental differences between sparse and dense matrices are discussed. The basic sparse matrix storage formats are considered: coordinate, compressed row storage (or compressed sparse rows, CRS) and compressed column storage (or compressed sparse columns, CCS) formats and also the storage formats for matrices of special form (symmetric, band, block matrices, etc.). The standard matrix algorithms are stated: matrix transposition, matrix-vector and matrix-matrix multiplication. These algorithms description is given taking into account matrix sparseness. The typical problems that arise when implementing obviously “dense” algorithms in “sparse” case and the methods for solving such problems are demonstrated.

# GUIDELINES

The standard operations with sparse matrices are considered. In the previous sections it was implicitly supposed that the matrices we worked with were dense.

The notion of sparse matrix might be defined in many ways and all of them consist in that there are “many” zero elements in the sparse matrix. As a rule, a matrix is said to be *sparse* if it contains *O*(*n*) nonzero elements. Otherwise, the matrix is called *dense*.

It is obvious that any sparse matrix can be processed as if it were dense, and vice versa. If the algorithms are properly implemented, the correct results will be obtained in both cases while the computational charges will differ considerably. Therefore, attributing the property of sparseness to a matrix is equivalent to contending that an algorithm exists which takes advantage of matrix sparseness and makes the matrix operations more efficient in comparison with the standard algorithms.

Many algorithms trivial for the cases of dense matrices require a more careful approach in the sparse case. A lot of algorithms that process sparse matrices can be divided into two stages: *symbolic* and *numerical*. At the symbolic stage, a *pattern* of the resulting matrix is formed (i.e. the locations of nonzero elements in the matrix structure are determined); at the numerical stage, the nonzero elements values of the resulting matrix are calculated.

The various sparse matrix storage formats exist. Some are designed for storing matrices of special form (e.g., band matrices), while others provide the work with matrices of general form.

The most obvious storage method for a certain sparse matrix is a *coordinate format*: only nonzero elements and their coordinates (rows and columns numbers) are stored. This format provides slow access to the matrix elements and consumes too much memory.

*Compressed sparse rows format* is one of the most widely used storage schemes for sparse matrices. This scheme has minimal memory requirements and at the same time is very useful for many important operations with sparse matrices: addition, multiplication, permutation of rows and columns, transposition, solving linear systems with sparse coefficient matrices by means of both direct and iterative methods, etc.

The compressed row storage format of the matrix *A* is specified by three one-dimensional arrays:

* *values* is an array that contains nonzero elements of the matrix *A*, which are listed by rows from the first to the last;
* *cols* is an array containing the numbers of the columns for the corresponding elements of the values array;
* *pointer* is an array of the pointers to the positions in two previous arrays, from which the description of every next row is started.

The compressed sparserows format ensures the efficient access to the matrix rows while access to the columns is problematic. Therefore, this storage method is preferable for the algorithms that contain mainly the row operations.

After considering the row storage format, the *compressed sparse columns format* becomes clear. In this case, the nonzero elements of the matrix *A* are listed in order of their appearance in the matrix columns, not in the rows. All the nonzero elements are stored by columns in an array *values*; the indices of the rows of the nonzero elements – in an array *rows*; the elements of an array *pointer* point to the positions, from which the description of every next column is started.

The column representations can be considered as the row representations of the transposed matrices. The compressed sparsecolumns format ensures the efficient access to the matrix columns while access to the rows is problematic. Therefore, this storage method is preferable for the algorithms that contain mainly the column operations. In case the processed matrix is symmetric, it is enough to store only its upper triangle. Moreover, any of the described formats may be used in this case.

Then, let us discuss the standard operations with the sparse matrices. Start with the simplest one: multiplication of the sparse matrix by a dense vector. The result of this operation is a dense vector, so the calculations can be made directly without the symbolic stage.

The transposition operation in case of the sparse matrices is implemented in more complicated way. The direct implementation of the algorithm results in quadratic algorithm complexity (with respect to the number of the nonzero elements) like in case of the dense matrices. An algorithm that has a linear complexity is considered.

Finally, discuss an algorithm for the matrices multiplication, i.e. calculating *C*=*AB*. Similar to the transposition case, the performance of such operation by the definition will be extremely inefficient. In order to multiply the sparse matrices efficiently, the following actions are necessary:

1. Implement transposition of sparse matrix and apply it to the matrix *B*.
2. Initialize a data structure for the matrix *C*, provide an opportunity to add elements to it.
3. Sequentially, multiply every row of the matrix *A* by every column of the matrix *BT* writing the derived results to *C* and forming its structure.

As follows from the description above, the basic operation is a scalar multiplication of two sparse vectors. Therefore, this operation should be implemented very carefully.

In conclusion, note that the considered basic algorithms demonstrate typical problems that arise when implementing obviously “dense” algorithms in “sparse” case.

# RECOMMENDATIONS for students

# References

1. Alan George, Joseph W. H. Liu. Computer solution of large sparse positive definite systems. Prentice-Hall, 1981.
2. Sergio Pissanetzky. Sparse matrix technology. Academic Press, 1984.
3. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. Numerical Recipes. The Art of Scientific Computing. Cambridge University Press, 2007.

# EXERCISES

1. Develop a sequential implementation for the algorithm of multiplication of sparse matrix by dense vector.
2. Based on the sequential program develop a parallel implementation using the OpenMP technology.
3. Estimate a cache misses number in the OpenMP-implementation when increasing threads number, in order to clarify the reasons for the non-linear scalability. Use Intel Parallel Studio XE tool.

# TEST QUESTIONS

1. Choose the most economical storage format for a tridiagonal matrix of size 1000×1000.
   1. The coordinate format
   2. The compressed sparse columns format
   3. + In the form of three arrays for three diagonals correspondingly.
2. Choose the most economical storage format for a matrix of size 1000×1000, which has a density of 1%.
   1. The coordinate format
   2. + The compressed sparse columns format
   3. The storage format for dense matrix
3. For the efficient implementation of multiplication operation for a sparse matrix by a dense vector the matrix should be stored in
   1. + the compressed sparse rows format
   2. the compressed sparse columns format
   3. the coordinate format
4. What kind of information is held in the array element **RowIndeх[i]** if the matrix is stored in the CRS format?
   1. The row number for the **i**-th nonzero matrix element
   2. + The start index for the elements of the **i**-th row in the array **Values**
   3. The start index for the elements of the **i**-th column in the array **Values**
5. What kind of information is held in the array element **ColIndeх[i]** if the matrix is stored in the CCS format?
   1. The row number for the **i**-th nonzero matrix element
   2. The start index for the elements of the **i**-th row in the array **Values**
   3. + The start index for the elements of the **i**-th column in the array **Values**
6. In what positions of the array **Values** the elements of the **i**-th row of the matrix are stored if the matrix is stored in the CRS format?
   1. From **Columns[i]** to **Columns[i + 1] - 1**
   2. + From **RowIndeх [i]** to **RowIndeх[i + 1] - 1**
   3. From **RowIndeх [i]** to **RowIndeх[i + 1]**
7. What algorithm scheme is the most efficient when multiplying two sparse matrices in the CRS format?
   1. Multiplication of А by В by definition
   2. +Transposition of В, then multiplication of А by BT by rows
   3. Transposition of A, then multiplication of AT by B by columns
8. What parallelization scheme could be applied for implementing the algorithm of matrices multiplication?
   1. + Loop parallelization by the rows of the matrix А
   2. Loop parallelization by the rows of the matrix В
   3. Loop parallelization by the columns of the matrix ВТ
9. What is the function of the symbolic stage of the matrices multiplication?
   1. + Formation of the pattern of the product, memory allocation for the resulting matrix
   2. Calculation of the elements of the product
   3. Counting the necessary memory volume for the resulting matrix without constructing its pattern