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

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 2. Gaussian elimination method for solving systems of general type*

Nizhni Novgorod

2014

# Objectives

The objective of the lecture is the investigation of the direct methods for solving linear equations systems with square matrix and also the research of the approaches to their efficient parallel implementation in shared memory systems.

# Abstract

Method of sequential elimination of the unknown variables (Gaussian method) for solving systems of *n* linear algebraic equations with a dense real matrix *A* and a right-hand side *b* (the case of the triangular or square matrix *A*) is considered. The modifications of the classical algorithms are stated. The estimates for the algorithms error that arises when calculating with an error are shown. The insufficient efficiency of using the classical algorithm in parallel computation systems is demonstrated. The idea of block data processing as a way to increase memory usage efficiency is described step-by-step.

# GUIDELINES

The methods for solving systems of *n* linear algebraic equations with a dense real matrix *A* and a right-hand side *b* are investigated. The problem of solving a linear equations system with the given matrix *A* and the vector *b* is considered to be a problem of searching a value of the unknown vector *x* such that all the equations of the system hold.

*The method of sequential elimination of the unknown variables*, or *the Gaussian method* is a base for all the algorithms studied in this lecture.

The Gaussian method is based on a possibility of linear equations transformations that do not change the solution of the given system (such transformations are called *equivalent*). These transformations include the following ones:

* multiplying any equation by a nonzero constant,
* the permutation of the equations,
* adding any system equation to other one.

First, examine the method with regard to the linear systems with the matrices of specific type – triangular matrices. For the sake of determinacy, let us assume that the matrix is upper triangular. The determination of the values of the unknowns is possible for such matrices. The value of the variable *xn* can be calculated from the last equation. After that, from the last but one equation the value of the variable *xn*−1 can be calculated, etc. The method complexity is *O*(*n*2).

This method can be parallelized (parallel computation of the sums while calculating the variables *xi*). However, such parallelization is inefficient. The given algorithm can be parallelized efficiently if several systems with the same matrix and several right-hand sizes are solved simultaneously. In this case, the block organization of the algorithm is possible: a portion of the matrix operations would amount to , where *N* is the number of the blocks. A solution error depends on the machine epsilon *εm*, the system size *n* and the matrix norm ||*A*||.

Let us examine now the Gaussian method for square matrices. It consists of the sequential execution of two stages. At the first stage, which is called *forward elimination phase*, the given system of the linear equations is reduced to the upper triangular form by eliminating sequentially the unknowns. During the *back substitution phase* (the second stage of the algorithm) the values of the unknowns are determined.

During the forward elimination phase of the Gaussian method, the row that is used for eliminating the unknowns is called the *pivot* one and its diagonal element is called the *pivot element*. The execution of the calculations is possible only when the pivot element does not equal zero. Moreover, if the value of the pivot element *aii* is small, then both multiplication and division of the rows by this element could lead to the accumulation of the computational errors and the computational instability of the algorithm.

It is possible to avoid such problem. In order to do this, two following steps should be made when executing each iteration of the forward elimination phase. First, it is necessary to detect the coefficient with the maximal absolute value in the column corresponding to the eliminated unknown. Then, chose the row which this coefficient belongs to as the pivot one. Such scheme of choosing the pivot value is called the *partial pivoting scheme*.

The *back substitution phase* of the algorithm is actually solving the system with the triangular matrix (has been already considered). The total execution time of the Gaussian method can be estimated as 2/3*n*3+*O*(*n*2) when *n* is great.

*LU-*decomposition (*LU-*factorization) is the representation of the matrix *A* in the form of *A*=*LU*, where *L* is a lower triangular matrix with the unit diagonal elements and *U* is an upper triangular matrix with the nonzero diagonal elements.

*LU-*decomposition algorithm is related to the Gaussian elimination method. The matrix *L* can be obtained as the lower triangular matrix of the Gaussian coefficients and the matrix *U* – as the upper triangular matrix derived as a result of executing the Gaussian method. The complexity of the *LU-*factorization execution is the same: 2/3*n*3+*O*(*n*2).

If the task is to solve several systems with the same coefficient matrix but with the different vectors of the free terms (the right-hand side of the linear system), the *LU*-factorization method is preferable because in this case there is no necessity to decompose the coefficient matrix many times. It is enough to store the derived triangular matrices in the memory and, after that, it is possible to obtain the solutions by means of forward and back substitution methods substituting the different vectors of the free terms. It allows us to reduce the computations volume considerably.

The solution error depends on the machine epsilon *εm*, the system size *n* and the matrices norms ||*L*|| and ||*U*||.

The method parallelization could be implemented in the following way. All the calculations are reduced to the independent one-type operations with the rows of the coefficient matrix of the linear equations system. As a result, the principle of parallelization by data distribution could be a base for parallel implementation of the Gaussian algorithm. All the calculations that include processing a group of the rows of the matrix *A* and the corresponding elements of the vector *b* could be considered as a subtask. In this case, applying the sequential scheme of data division for parallel solving the linear equations systems leads to an unequal computational load among the processors: as the unknowns for the most threads will be eliminated, all required calculations will be completed and the threads will happen to be idle. The possible solution for the problem of the calculations balance may consist in using the blocked-striped scheme of data distribution among the subtasks.

The results of the experiments demonstrate that, when the number of the threads is *p*≥3, the speedup is approximately 1.5 and it depends neither on the number of the threads, nor on the matrix size, although we could expect better results according to the preliminary estimates (an exception is the case of *N*=1000). The absence of considerable speedup is caused by the following phenomenon.

First, if *N*=1000, the matrix size makes it possible to load the matrix to the processor cache entirely and to use the high-speed memory for executing the operations. Meantime, if *N*>1000, the matrix does not fit in the cache-memory entirely, so the cache misses number increases and this leads to frequent accessing the main memory that is slow enough. The cache memory may be used efficiently if the block data division is used (this will be shown later).

Secondly, compiler optimizes the simpler sequential program more efficiently than the more complicated parallel one. In some cases, one-thread version of the parallel program will work 1.5-2 times slower than its sequential counterpart. That’s why hereinafter we will estimate the speedup of the parallel program executed on *p* threads relative to the same program executed on one thread.

The disadvantage of the examined standard *LU*-decomposition algorithm is conditioned by the fact that the algorithm computations conform poorly to the rules of using the *cache-memory*. The cache-memory is the extra high-speed computer memory that is used for storing the copies of the most frequently used main memory areas. The result of the cache-memory usage will be observable if the executed calculations use the same data many times and access the memory elements with the sequentially increasing addresses.

In the considered *LU*-decomposition algorithm the data is located in memory by rows, while the calculations are executed by columns. This results in low efficiency of cache-memory usage. A possible way to improve the situation is the consolidation of the computational operations, leading to the sequential processing of some rectangular submatrices of the matrix *A*.

*LU*-decomposition may be organized so that the matrix operations would be basic operations. Note that implementation of the matrix operations makes the efficient cache-memory usage possible. Let us represent the matrix *A* in a block form and then execute the block decomposition. One step of the block algorithm consists in the following:

1. The execution of common *LU*–decomposition for a small submatrix of size r×r.
2. Solving two systems with the triangular matrices and many right-hand sides.
3. The calculation of the reduced matrix of smaller size (n−r)×(n−r) as a result of matrix multiplication.

After that, the algorithm is applied to the reduced matrix.

As well as the other investigated decomposition procedures, the given scheme requires 2/3*n*3+*O*(*n*2) operations. A portion of the matrix operations may be estimated as 1−1/*N*2, where *N* is the number of the blocks.

The block *LU*-decomposition algorithm should be parallelized on the level of matrix operations.

The results indicate the good (almost linear) scalability of the block algorithm in the opposite to the standard version of the Gaussian method.

# RECOMMENDATIONS for students

The books [1], [2] include the description of the classical variants of the algorithms. Also the problems of stability of the obtained solutions, when calculating with errors, are studied there. The dependence of the derived solutions accuracy on the matrix conditionality of the linear system is demonstrated.

The block variants of the algorithms are stated in the books [3], [4]. Taking into account the matrix and scalar operations, the complexity estimates are given. Every algorithm description is followed by the pseudocode. The organization of the parallel computations for shared memory is considered.

The implementations of the parallel algorithms on pseudocode (for the case of distributed memory) are stated in [5]. A concrete realization of the parallel algorithms on C/C++ could be found in [6].

# References

1. David R. Kincaid and E. Ward Cheney. Numerical analysis: mathematics of scientific computing. Brooks/Cole Publishing Company, 1991.
2. Richard L. Burden, J. Douglas Faires. Numerical Analysis. Brooks Cole, 2000.
3. Gene H. Golub, Charles F. Van Loan. Matrix Computations. The John Hopkins University Press, 1996.
4. James W. Demmel. Applied Numerical Linear Algebra. SIAM, 1997.
5. M. Quinn. Parallel programming in C with MPI and OpenMP. McGraw-Hill, 2004.
6. 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. Implement the block algorithm for solving the linear equations system with a triangular matrix and several right-hand sides *LX*=*B*.
2. Implement *LU*-decomposition for a square matrix.
3. Implement the parallel block *LU*-decomposition for a square matrix.

# TEST QUESTIONS

1. What is the complexity order of the Gaussian elimination method when applying to triangular matrix?
   1. O(n)
   2. +O(n2)
   3. O(n3)
2. Is it possible to parallelize the Gaussian method efficiently when solving system with triangular matrix?
   1. Yes, it is possible
   2. +Yes, but only if solving series of the systems with the same matrix and different right-hand sides
   3. No, it is not
3. What is the complexity order of the Gaussian elimination method when applying to square matrix?
   1. O(n)
   2. O(n2)
   3. +O(n3)
4. The connection between the Gaussian elimination method and the *LU*-decomposition consists in the following:
   1. The matrix *L* can be obtained as the lower triangular matrix of the Gaussian coefficients and the matrix *U* – as the upper triangular matrix derived as a result of executing the elimination method
   2. The matrix *L* can be obtained as the lower triangular matrix of the Gaussian coefficients and the matrix *U* – as the upper triangular matrix derived as a result of transposition of the matrix *L*
   3. The matrix *L* can be obtained as the result of the elimination method work and the matrix *U* – as the upper triangular matrix derived as a result of transposition of the matrix *L*
5. The parallel *LU*-decomposition algorithm, when using non-block parallelization and working with the matrices of great size,
   1. demonstrates low efficiency because of the fast execution of the sequential algorithm
   2. + demonstrates low efficiency because of the great number of the cache misses in the parallel algorithm
   3. demonstrates high efficiency
6. Which of the three variants of the *LU*-decomposition (by row, by column, compact scheme) is preferable for the implementation (matrix is stored by rows)?
   1. +By column, since all the elements of the rows from the *j*-th to the *n*-th are used when calculating the *j*-th column and the access to the matrix row in C language is organized efficiently.
   2. By row, since all the elements of the columns from the first to the *i*-th are used when calculating the *i*-th row and the access to the matrix column in C language is organized efficiently.
   3. The compact scheme since it has the smaller complexity.
7. What is the relation between the error of the Gaussian method without choosing the pivot element and the error of the Gaussian method with choosing the pivot element by column?
   1. The errors are the same and they are comparable to rounding error
   2. The error of the common Gaussian method is smaller than the error of the method with choosing the pivot element
   3. + The error of the Gaussian method with choosing the pivot element is smaller than of the common method.
8. What is the relation between the error of the Gaussian method without choosing the pivot element and the error of the Gaussian method with choosing the pivot element by column and by row?
   1. The errors are the same and they are comparable to rounding error
   2. The error of the common Gaussian method is smaller than of the method with choosing the pivot element
   3. + The error of the Gaussian method with choosing the pivot element is smaller than of the common method.
9. Is the block *LU*-decomposition algorithm more efficient than the common algorithm?
   1. No, the method has the same efficiency because their complexities are equal
   2. No, the common algorithm is more efficient in view of smaller complexity
   3. Yes, the block algorithm is more efficient in view of smaller complexity
   4. + Yes, the block algorithm is more efficient in view of proper cache-memory usage although their complexities are the same.
10. What stage of the parallel block *LU*-decomposition algorithm is the most time-consuming and thus it determines the total efficiency of the algorithm?
    1. Execution of the *LU*-decomposition for submatrix
    2. Solving the system with the triangular matrix and many right-hand sides
    3. + Calculation of the reduced matrix