Recursive Algorithms for Sparse Matrices in Domain

1st Gennadi Malaschonok
Tambov State University
Tambov, Russia
malaschonok@gmail.com

2nd Evgeni Ilchenko
Tambov State University
Tambov, Russia
ilchenkoea@gmail.com

***Abstract*—We give an overview of the theoretical results for matrix block-recursive algorithms in commutative domains and present the results of experiments that we conducted with new parallel programs based on these algorithms on a supercomputer MVS-10P at the Joint Supercomputer Center of the Russian Academy of Science. To demonstrate a scalability of these programs we measure the running time of the program for a different number of processors and plot the graphs of efficiency factor. Also we present the main application areas in which such parallel algorithms are used. It is concluded that this class of algorithms allows to obtain efficient parallel programs on clusters with distributed memory.**

***Index Terms*—block-recursive matrix algorithms, commutative domain, factorization of matrices, matrix inversion, generalized inverses, distributed memory**

I. Introduction

J. Dongarra at his talk at International Congress ICMS-2016 [1] put attansion on the several difficult challenges. He noted that the task of managing calculations on a cluster with distributed memory for algorithms with sparse matrices is today one of these the most difficult challenges.

We have to add also one more problem. It is a high computational complexity, that can be connected with the type of the basic algebra: you can take a matrix over field or over commutative ring.

For sparse matrices, it is not true that all computations over polynomials or integers can be effectivelly reduced due to the technic of modular computations. It was proved in theoretical investigatios of computational complexity for some algorithms with sparse matrices in [2]. Below we give experiments with large sparse matrices, which confirm these theoretical reults.

We consider the class of block-recursive matrix algorithms. The most famous of them are standard and Strassen's block matrix multiplication, Strassen's block-matrix inversion [3].

Block-recursive algorithms were not so important as long as the calculations were performed on computers with shared memory. Only in the nineties it became clear that block-recursive matrix algorithms are required to operate with sparse large matrices on a supercomputer with distributed memory.

Note that the generalization of Strassens matrix inversion algorithm [3] with additional permutations of rows and columns by J. Bunch and J. Hopkroft [4] is not a block-recursive algorithm.

This work was completed with the support of RFBR grant No 16-07-00420.

The block recursive algorithm for the solution of systems of linear equations and for adjoint matrix computation which is some generalisation of Strassen's inversion in commutative domains was suggested in the papers [8], [9] and [11]. See also at the book [10]. However, in all these algorithms, except matrix multiplication, a very strong restriction are imposed on the matrix: the leading corner minors should not be zero.

This restriction was removed later. The algorithm that computes the adjoint matrix, the echelon form, and the kernel of the matrix operator for the commutative domains was proposed in [12]. The block-recursive algorithm for the Bruhat decomposition and the LEU decomposition for the matrix over the field was obtained in [13], and these algorithms were generalized for the matrices over commutative domains in [15] and in [16].

In this article we review the main achievements in this class of algorithms and present the results of experiments that we conducted with these algorithms on a supercomputer MVS-10P at the Joint Supercomputer Center of the Russian Academy of Science.

In the next section, we present the main application areas in which such algorithms are used.

II. Some important areas for applications of sparse matrices algorithms

A. Computations of functions of electronic circuits

The behavior of electronic circuits can be described by Kirchhoff's laws. The three basic approaches in this theory are direct current, constant frequency current and a current that varies with time. All these cases require the compilation and solution of sparse systems of equations (numerical, polynomial or differential). The solution of such differential equations by the Laplace method also leads to the solution of polynomial systems of equations [17].

B. Control systems

In 1967 Howard H. Rosenbrock introduced a useful state-space representation and transfer function matrix form for control systems, which is known as the Rosenbrock System Matrix [18]. Since that time, the properties of the matrix of polynomials being intensively studied in the literature of linear control systems.

C. Computation of Gröbner bases

Another important application is the calculation of Gröbner bases. A matrix composed of Buchberger S-polynomials is a strongly sparse matrix. Reduction of the polynomial system is performed when calculating the echelon and diagonal forms of this matrix. The algorithm F4 [19] was the first such matrix algorithm.

D. Solving ODE's and PDEs.

Solving ODE's and PDE's is often based on solution of leanear systems with sparse matrices over numbers or over polynomials. One of the important class of sparse matrix is called quasiseparable. Any submatrix of quasiseparable matrix entirely below or above the main diagonal has small rank. These quasiseparable matrices arise naturally in solving PDEs for particle interaction with the Fast Multi-pole Method (FMM). The efficiency of application of the block-recursive algorithm of the Bruhat decomposition to the quasiseparable matrices is studied in [21].

III. Development of the recursive matrix agorithms in integral domain

We can trace how developed the matrix recursive agorithms in integral domain, which eventually led to the creation of modern algorithms. There are several separate periods.

A. Algorithms for solution of a system of linear equations of size n in an integral domain, which served as the basis for recursive algorithms

(1983) Forward and backward algorithm ($\sim n^3$) [5].
(1989) One pass algorithm ($\sim \frac{2}{3}n^3$) [6].
(1995) Combined algoritm with upper left block of size r ($\sim \frac{7}{12}n^3$ for $r = \frac{n}{2}$) [7].
Really, this was already the first step of a recursive algorithm. It was first discovered that when the matrix is divided into equal four blocks ($r = \frac{n}{2}$), the least computational complexity is achieved. Consequently, further dichotomous division of blocks can give the best algorithm. It remains to prove several determinant identities that would allow us to do recursive calculations.

B. Recursive algorithms for solution of a system of linear equations and for adjoint matrix computation in an integral domain without permutations

(1997) Recursive algorithm for solution of a system of linear equations [8].
(2000) Adjoint matrix computation (with 6 levels) [9].
(2006) Adjoint matrix computation alternative algorithm (with 5 levels) [11].
Now it remained to solve the problem of permutation of blocks and ensure the fulfillment of determinant identities.

C. Main recursive algorithms for matrices in a domain

(2008) Computation of adjoint and inverse matrices and the operator kernel in a domain [12].
(2010) Bruhat and LEU decompositions in a feild [13].
(2012) Bruhat and LDU decompositions in a domain [14], [15].
Recursive algorithms for sparse matrices in commutative domains with the complexity of matrix multiplication are obtained. The complexity of computing the matrix product for matrices of size n we denote by $\sim n^\beta$.

D. New achivements and new applications

(2013) It is proved that the LEU algorithm has the complexity $O(n^2 r^{\beta-2})$ for rank r matrices [20].
(2015) New algorithms for Bruhat and LDU decompositions in a domain (alternative algorithm) [16].
(2017) It is proved that the LEU algorithm has the complexity $O(n^2 s^{\beta-2})$ for quasiseparable matrix. A matrix is called quasiseparable if any it's submatrix which entirely disposed below or above the main diagonal has small rank s, $s << n$ [21].

IV. Recursive standard and strassen's matrix multiplication

The graph of recursive algorithm for standard matrix multiplication is shown at Figure 1.

$$\begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} \times \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix} + \begin{pmatrix} C_0 & C_1 \\ C_2 & C_3 \end{pmatrix} = \begin{pmatrix} D_0 & D_1 \\ D_2 & D_3 \end{pmatrix}$$

Number of operations for the standard algorithm is $\sim n^3$.
The graph of the Strassen multiplication algorithm can be easily represented similarly. The number of operations for this algorithm is $\sim n^{\log_2 7}$.

The algorithm for multiplying matrices on leaf tops should take into account the sparse matrix structure and compact storage form.

We note that there exists a boundary with respect to the density of the matrix, which separates the region of applicability of the Strassen multiplication. If the density of the matrix is below this boundary, then only standard multiplication is effective (see details in [2]).

V. Recursive Strassen's matrix inversion

If $\mathcal{A} = \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix}$, $\det(\mathcal{A}) \neq 0$ and $\det(A_0) \neq 0$ then

$$\mathcal{A}^{-1} = \begin{pmatrix} \boldsymbol{I} & -A_0^{-1}A_1 \\ 0 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & 0 \\ 0 & (A_3 - A_2A_0^{-1}A_1)^{-1} \end{pmatrix} \times$$

$$\begin{pmatrix} \boldsymbol{I} & 0 \\ -A_2 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} A_0^{-1} & 0 \\ 0 & \boldsymbol{I} \end{pmatrix} = \begin{pmatrix} M_1M_5 - M_0 & M_1M_4 \\ M_5 & M_4 \end{pmatrix}.$$

We have denoted here $M_0 = -A_0^{-1}$, $M_1 = M_0A_1$, $M_2 = A_2M_0$, $M_3 = M_2A_1$, $M_4 = (A_3 + M_3)^{-1}$, $M_5 = -M_4M_2$.

The graph of recursive Strassen's matrix inversion is shown on Figure 2.

Fig. 1. Recursive standard matrix multiplication.

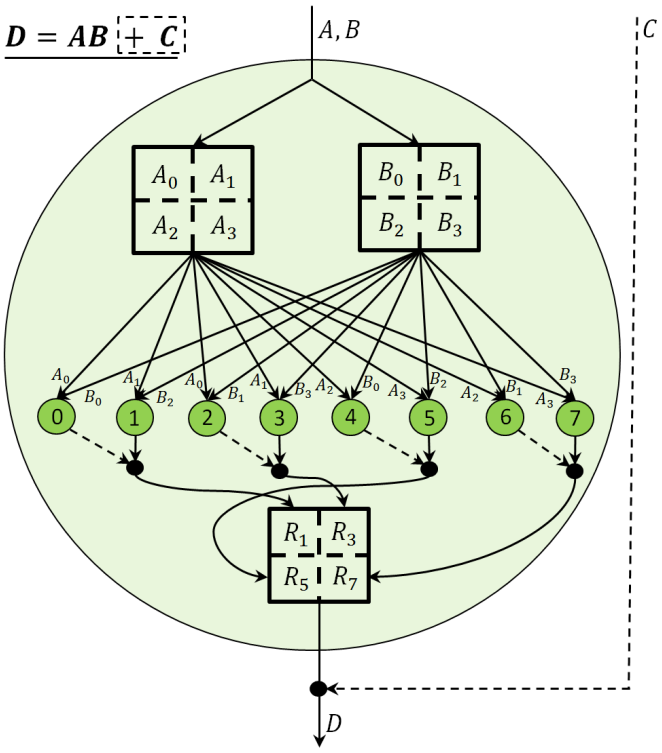
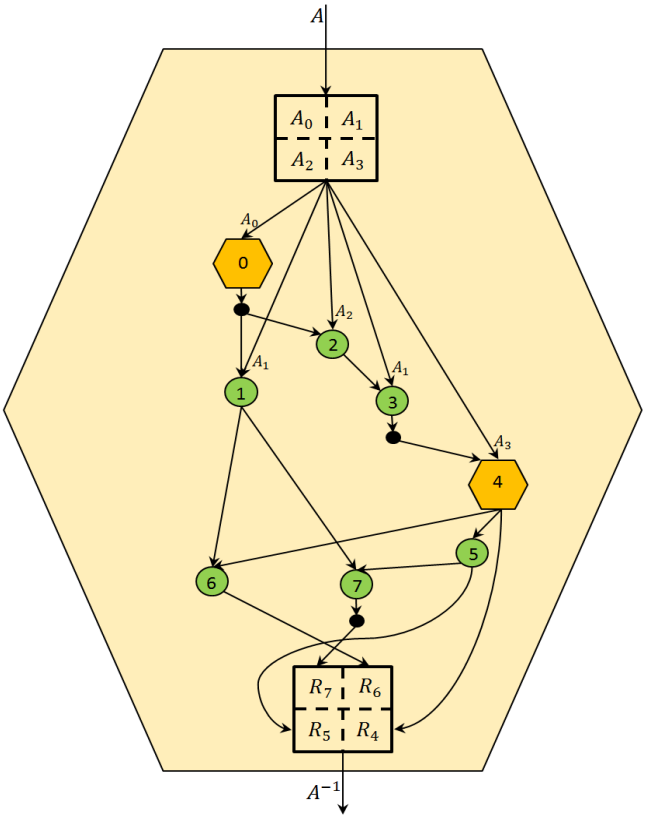


Fig. 2. The graph of recursive Strassen's matrix inversion



VI. Recursive inversion of triangular matrix

If $\mathcal{A} = \begin{pmatrix} A & 0 \\ B & C \end{pmatrix}$ is triangular matrix of order 2^k and $\det(\mathcal{A}) \neq 0$ then

$$\mathcal{A}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ -C^{-1}BA^{-1} & C^{-1} \end{pmatrix}.$$

VII. Recursive Cholesky decomposition

Let $\mathcal{A} = \begin{pmatrix} A_1 & A_2 \\ A_2 & A_3 \end{pmatrix}$ be a positive definite symmetric matrix and $H = \begin{pmatrix} B & 0 \\ C & D \end{pmatrix}$ be a low triangle matrix with the property $\mathcal{A} = HH^T$. The mapping

$$Chol: \ R^{n\times n} \to (R^{n\times n}, R^{n\times n}),$$
$$Chol(\mathcal{A}) = (H, H^{-1})$$

is called an *Cholesky decomposition*. Let $n = 2^k$, then you can used following recursive algorithm.

1) : Let $Chol(A_1) = (B, B^{-1})$.

We can compute C=$A_2^T(B^{-1})^T$ and $F = A_3 - CC^T$

2) : Let $Chol(F) = (D, D^{-1})$.

Then $H = \begin{pmatrix} B & 0 \\ C & D \end{pmatrix}$ and $H^{-1} = \begin{pmatrix} B^{-1} & 0 \\ -D^{-1}CB^{-1} & D^{-1} \end{pmatrix}$.

VIII. Recursive computation of the adjoint matrix, kernel and determinant

We consider matrices over a commutative domain.

Semigroup P_n is formed by $n \times n$ matrices, which have the number of unit elements coincides with its rank, and the remaining elements are zero. The semigroup D_n is formed by the diagonal matrices: $D_n \subset P_n$, $|D_n| = 2^n$. The identity matrix $\mathbf{I}$ is a unit in D_n and in P_n. For each matrix $E \in P_n$ we define diagonal matrices $I_E = EE^T \in D_n, J_E = E^TE \in D_n$. Also, we used the involution fanction on $D_n : \bar{I} = \mathbf{I} - I$, $\forall I \in D_n$ with the property $\bar{\bar{I}} = I$.

For the matrix E, the matrix $\bar{I}_E$ is a left annihilator, and the matrix $\bar{J}_E$ is a right annihilator. So we can denote the set of echelon matrix of order n: $S_n = \{S \ | \exists E \in P_n, \ \exists d \in R \setminus 0 : \ S = I_E S, \ dE = SJ_E\}$. In other words, dE ($E \in P_n, \ d \in R \setminus 0$) is a block of echelon matrix $S \in S_n$ with $\text{rank } E = \text{rank } S$, such that the sets of zero rows of the matrices S and E coincide, and each nonzero column of the matrix dE coincides with the same column of the matrix S. We write: $E = E_S, S \in S_n$.

Below we will use such notation for any matrix $S_{ij} \in S_n$ and $E_{ij} = (E_{ij})_{S_{ij}}$:

$$I_{ij} = E_{ij}E_{ij}^T, \bar{I}_{ij} = \boldsymbol{I} - I_{ij}, Y_{ij} = E_{ij}^T S_{ij} - d_{ij}\boldsymbol{I}, \quad i,j \in 1,2.$$

Definition: The mapping

$$A_{ext}: \ R^{n\times n} \times (R\backslash 0) \to (R^{n\times n})^3 \times (R\backslash 0),$$
$$A_{ext}(M, d_0) = (A, S, E_S, d)$$

for $n = 2^k$ is called an *extended adjoint mapping* of the pair (M, d_0) if it is defined recursively as follows.

For $M = 0$ we define $A_{ext}(0, d_0) = (d_0 I, 0, 0, d_0)$.

For $k = 0$ and $M = a \neq 0$ we define $A_{ext}(a, d_0) = (d_0, a, a, a)$.

In all other cases, we split the matrix M into four equal blocks $M = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}$

1) : Let $A_{ext}(M_{11}, d_0) = (A_{11}, S_{11}, E_{11}, d_{11})$.

We denote $M_{12}^1 = A_{11}M_{12}/d_0$, $M_{21}^1 = -M_{21}Y_{11}/d_0$, $M_{22}^1 = M_{22}d_{11} - M_{21}E_{11}^T M_{12}^1/d_0$.

2) : Let $A_{ext}(\bar{I}_{11}M_{12}^1, d_{11}) = (A_{12}, S_{12}, E_{12}, d_{12})$.

3) : Let $A_{ext}(M_{21}^1, d_{11}) = (A_{21}, S_{21}, E_{21}, d_{21})$.

We denote $M_{22}^2 = -A_{21}M_{22}^1 Y_{12}/(d_{11})^2$, $d_s = d_{21}d_{12}/d_{11}$.

4) : Let $A_{ext}(\bar{I}_{21}M_{22}^2, d_s) = (A_{22}, S_{22}, E_{22}, d_{22})$.

We denote $M_{11}^2 = -S_{11}Y_{21}/d_{11}$, $M_{12}^2 =$

$$((\frac{S_{11}E_{21}^T A_{21}}{d_{11}}M_{22}^1 - I_{11}M_{12}^1 d_{21})/d_{11} * Y_{12} + S_{12}d_{21})/d_{11},$$

$$M_{12}^3 = -M_{12}^2 Y_{22}/d_s,\ M_{22}^3 = S_{22} - I_{21}M_{22}^2 Y_{22}/d_s,$$

$$A^1 = A_{12}A_{11},\ L = (A^1 - (I_{11}M_{12}^1 E_{12}^T A^1)/d_{11})/d_{11} * d_{22},$$

$$A^2 = A_{22}A_{21},\ P = (A^2 - (I_{21}M_{22}^2 E_{22}^T A^2)/d_s)/d_{21},$$

$$F = -((S_{11}E_{21}^T A_{21})/d_{11} * d_{22} + (M_{12}^2 E_{22}^T A^2)/d_s)/d_{21},$$
$$G = -((M_{21}E_{11}^T A_{11})/d_0 * d_{12} + (M_{22}^1 E_{12}^T A^1)/d_{11})/d_{11},$$

$$A = \begin{pmatrix} (L + FG)/d_{12} & F \\ (PG)/d_{12} & P \end{pmatrix},\ S = \begin{pmatrix} (M_{11}^2 d_{22})/d_{21} & M_{12}^3 \\ (S_{21}d_{22})/d_{21} & M_{22}^3 \end{pmatrix},$$

$$E = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix},\ d = d_{22}.$$

Then $A_{ext}(M, d_0) = (A, S, E, d)$.

Sentence. The map $A_{ext}(M, 1) = (A, S, E, d)$ defines an extended adjoint matrix A, an echelon matrix S, and a matrix E_S such that $AM = S$ and $dE = SJ_E$ [12]. The graph of extended adjoint map is shown at Figure 3.

IX. Results of experiments with matrix recursive algorithms on a cluster with distributed memory

The block-recursive matrix algorithms require a special approachs to managing parallel programs. One approach to the cluster computations management is a scheme with one dispatcher.

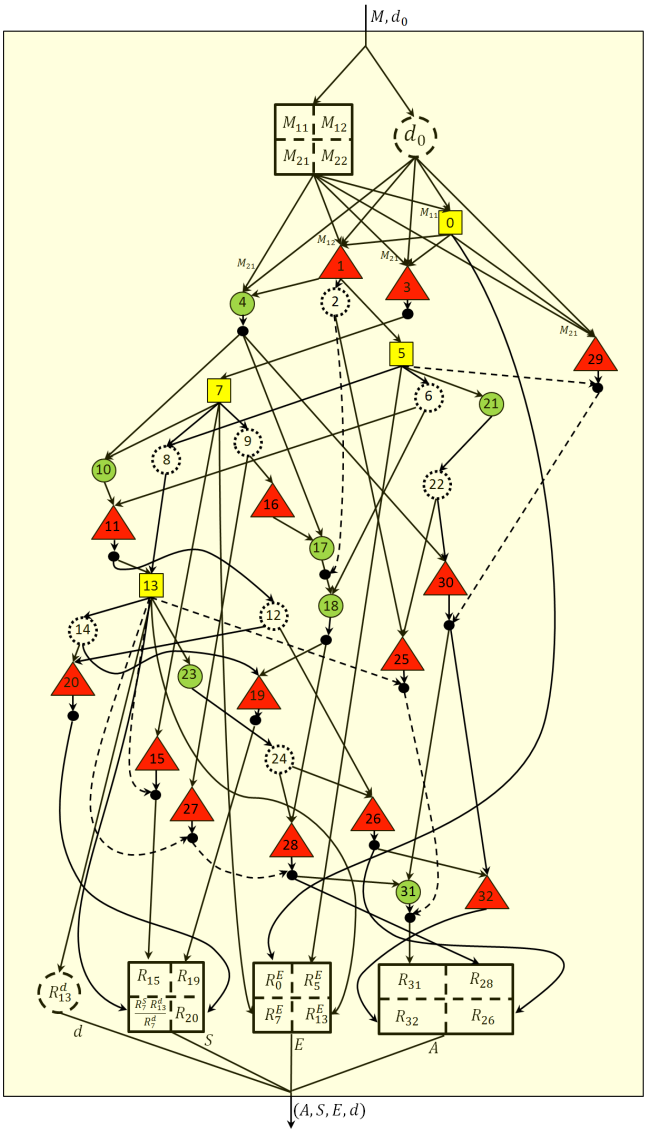
We consider another scheme of cluster menagement. It is a scheme with multidispatching, when each involved computing module has its own dispatch thread and several processing threads. Each processor, along with its subtask, receives a list of slave processors. During the computation, this list changes when new processors are added or when some of these processors complete their subtasks [22], [23].

A. Scalability

We have done the experiments using the supercomputer MVS-10P based on RSC Tornado architecture. It is a 10 Petaflops supercomputing system at the Joint Supercomputer Center (JSCC) of the Russian Academy of Science (RAS).

We demonstrate the results of experiments with parallel programms on the base of multidispatching. We demonstrate

Fig. 3. The graph of recursive computation of adjoint matrix and kernel.



a scalability of these programs. To do this, we plot the graph of efficiency factor.

For an "ideal parallel program" the product of the time t_n for solving the computational problem by the number n of cores in the computational cluster must remain constant:

$$t_n n = const.$$

So the value

$$f = \frac{t_n n}{t_k k} 100\%$$

can be taken as the "efficiency factor" of the n-cores with respect to the k-cores.

In order to investigate how the efficiency factor changes with increasing number of cores for a given program, it is possible to conduct a series of experiments with different number of cores in the cluster. If the program has this

coefficient above 50% for some range of number of cores, then we conside that it has good scalability in this range. We conducted several series of experiments and obtained graphs that show how the efficiency factor varies. In all experiments, except Strassen's matrix inversion, we took matrices with 15 bit integers.

For the algorithm of recursive Strassen's matrix inversion, we took a dense matrix with double-precision numbers. Figures 6 and 7 show the results of experiments with dence matrices of sizes 8000x8000 and 16000x16000, correspondingly. For a cluster having 200 cores, the efficiency coefficient is equal to 51% for a matrix size of 8000x8000 and 73% for a matrix size of 16000x16000.

In Figure 8, the efficiency is shown for a recursive algorithm for computing the adjoint matrix and the kernel, when the number of cores in a cluster changes from 8 to 400. We took arbitrary dense matrices with size 8000x8000. For a cluster having 200 cores, the efficiency coefficient is equal to 66%. Efficiency coefficient drops to 44% when the number of cores reaches 400.

Figures 4– 7 demonstrate that the larger the size of the matrix, the better the efficiency coefficient remains with increasing cluster sizes.

B. Comparison of the calculation time

The next three figures show how the calculation time varies with the growth of the matrix sizes for different algorithms and for different sparseness of the matrices.

We investigate the algorithm of computing the adjoint matrix and the kernel (see the algorithm in Figure 3). Experiments with two different versions of this algorithm are compared in Figures 9 and 10. These experiments were carried out for integer matrices on a cluster with 100 cores. We compared a simple algorithm and an algorithm in which the Chinese remainder theorem (CRT) was applied. It is well known that the application of the Chinese remainder theorem makes it possible to reduce the number of operations in such algorithms by $\sim n$ times, where n is the size of the matrix, if the standard algorithm for multiplying integers is used. This is true for dense matrices.

Figure 9 shows the results of experiments for dense matrices. For a matrix of size 2500x2500, the CRT algorithm is about twice as fast as a simple algorithm.

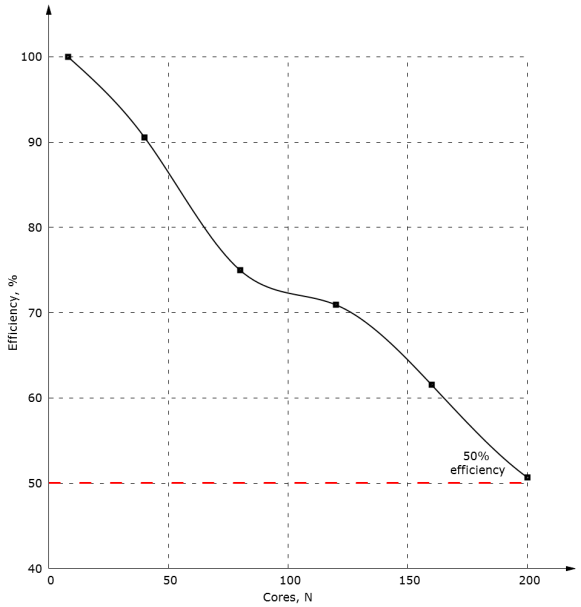
Figure 10 shows the results of experiments for sparse matrices that have a density of 1 percent. For a matrix of size 2500x2500, the CRT algorithm is approximately twice as slow as the simple algorithm.

We see that for very sparse matrices, the CRT algorithm should not be used. The explanation is simple. For sparse matrices, only a small number of matrix elements need be calculated, since many elements become zero during the computation.

C. Comparison with sequential programs in Mathematica and MAPLE

An experimental comparison of the sequential recursive algorithm for computing the adjoint matrix with similar programs in Mathematica and MAPLE systems is shown in Figure 11.

Fig. 4. Recursive Strassen matrix inversion, size=8000x8000.



We compared our algorithms with Mathematica 11 and MAPLE 2015. For comparison, we took random dense integer matrices and performed calculations with identical matrices in 4 programs. The best time of calculations was demonstrated by a sequential program. For example, for 600x600 matrices, it is twice as fast as Mathematica 11 and 7 times faster than MAPLE 2015. A slightly worse calculation time was shown by the fourth program. This is the parallel program that was run on a single processor.

X. Conclusion

The algorithms we discussed were used in the cloud computing system of the computer algebra Mathpartner. You can used this cloud system on the websites mathpar.cloud.unihub.ru and mathpartner.ukma.edu.ua.

Functions that correspond to these algorithms can be called by the operators *adjoint*, *kernel*, *det*, *LDU*, *BruhatDecomposition*.

To create a public cluster where you can organize multiprocessor computations of such matrix functions, requires a special international co-operation program. Such a program could be supported by the European Union or other form of international cooperation.

XI. Acknowledgment

The authors are grateful to the Ivannikov Institute for System Programming of the of the Russian Academy of Science for hosting the cloud computer algebra system Math Partner and to the Joint Supercomputer Center of the Russian Academy of Science for providing the ability to perform calculations on the supercomputer MVS-10P.

Fig. 5. Recursive Strassen matrix inversion, size=16000x16000.

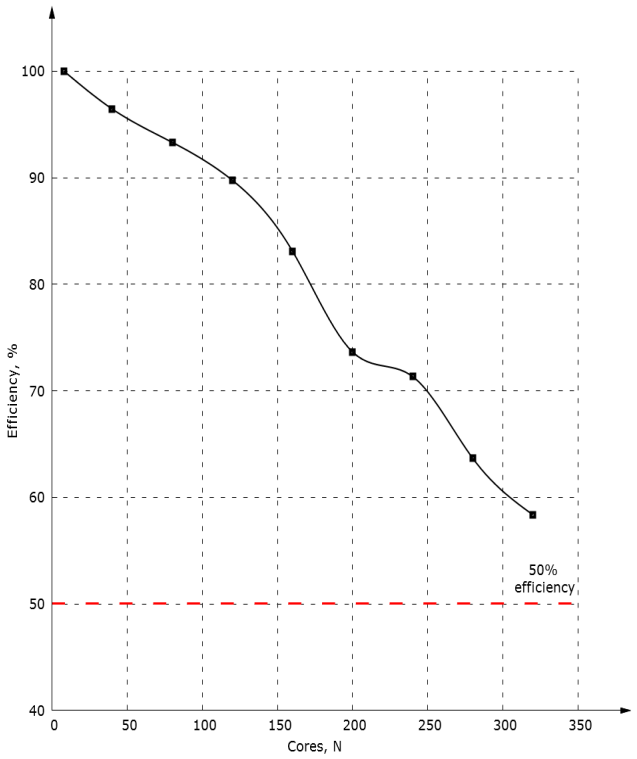


Fig. 6. Recursive algorithm for compution of adjoint matrix and kernel, size=8000x8000.

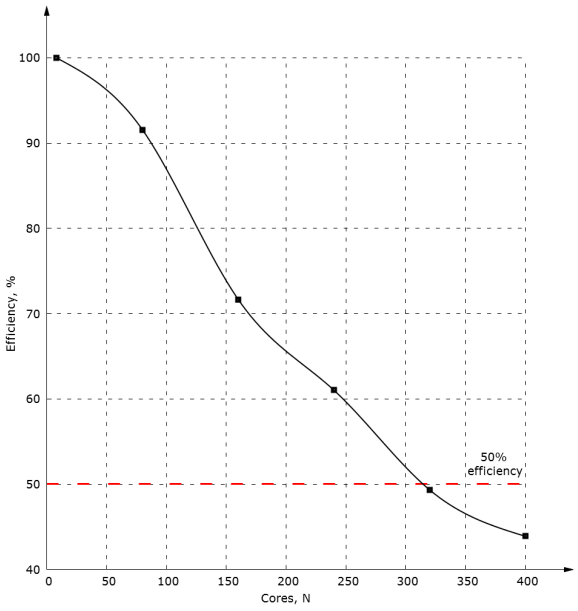


Fig. 7. Recursive algorithm for compution of adjoint matrix and kernel on the cluster with 100 cores, comparison of a simple algorithm and CPT algorithm for dense matrices.

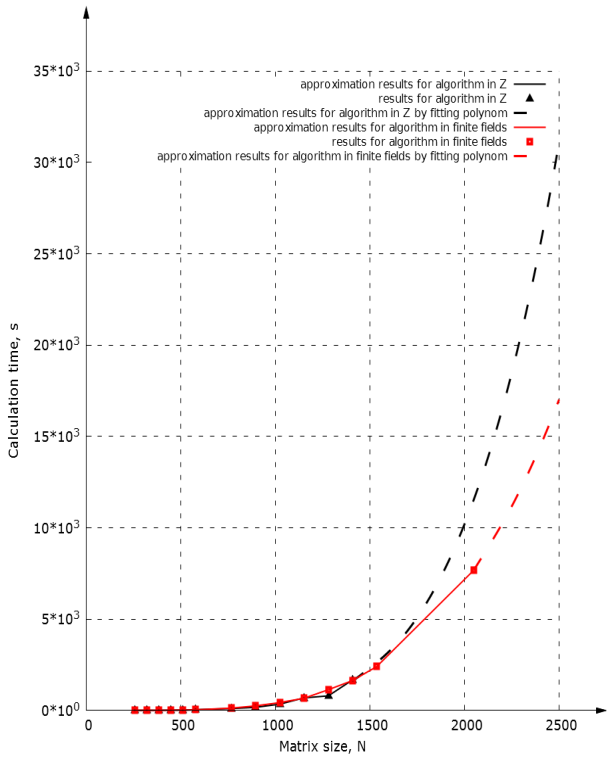


Fig. 8. Comparison of a simple and CPT algorithms of compution of adjoint matrix and kernel for sparse matrices with 1% of dencity.

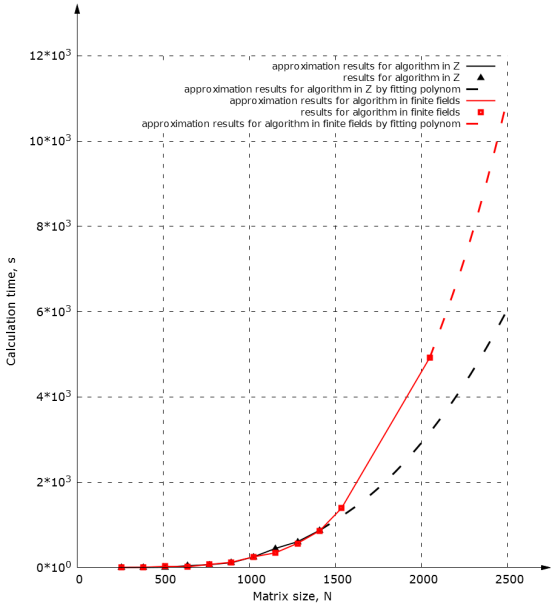
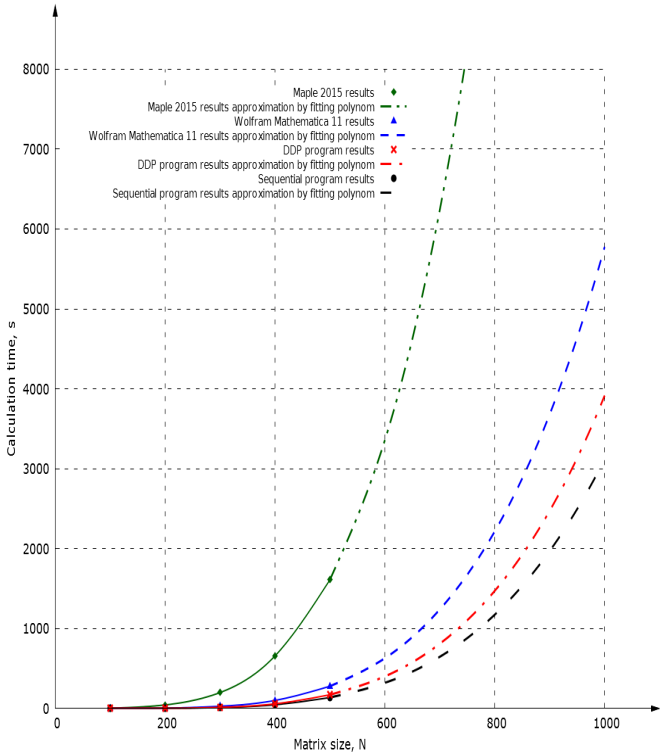


Fig. 9. Comparison of the sequential recursive algorithm for computing the adjoint matrix and the kernel with in Mathematica and MAPLE.



REFERENCES

[1] Dongarra J. *With Extrim Scale Computing the Rules Have Changed.* In Mathematical Software. ICMS 2016, 5th International Congress, Proceedings (G.-M. Greuel, T. Koch, P. Paule, A. Sommese, eds.), Springer, LNCS, volume 9725, pp. 3-8, (2016)

[2] Malashonok, G.I., Valeev, Y.D. & Lapaev, A.O. On the choice of a multiplication algorithm for polynomials and polynomial matrices. J Math Sci. V. 168, No. 3, 398416. DOI 10.1007/s10958-010-9992-z (2010)

[3] *Strassen V. Gaussian Elimination is not optimal.* Numerische Mathematik. V. 13, Issue 4, 354–356 (1969)

[4] Bunch J., Hopkroft J. *Triangular factorization and inversion by fast matrix multiplication.* Mat. Comp. V. 28, 231-236 (1974)

[5] Malaschonok G.I. *Solution of a system of linear equations in an integral domain,* rm Zh. Vychisl. Mat. i Mat. Fiz. V.23, No. 6, 1983, 1497-1500, Engl. transl.: USSR J. of Comput. Math. and Math. Phys., V.23, No. 6, 497-1500. (1983)

[6] G.I. Malaschonok. *Algorithms for the solution of systems of linear equations in commutative rings.* Effective methods in Algebraic Geometry, Progr. Math., V. 94, Birkhauser Boston, Boston, MA, 1991, 289-298. (1991) rm

[7] G.I. Malaschonok. *Algorithms for computing determinants in commutative rings.* Diskret. Mat., 1995, Vol. 7, No. 4, 68-76. Engl. transl.: Discrete Math. Appl., Vol. 5, No. 6, 557-566 (1995).

[8] Malaschonok G. *Recursive Method for the Solution of Systems of Linear Equations.* Computational Mathematics. A. Sydow Ed, Proceedings of the 15th IMACS World Congress, Vol. I, Berlin, August 1997), Wissenschaft & Technik Verlag, Berlin, 475-480. (1997)

[9] Malaschonok G. *Effective Matrix Methods in Commutative Domains*, Formal Power Series and Algebraic Combinatorics, Springer, Berlin, 506-517. (2000)

[10] Malaschonok G. *Matrix computational methods in commutative rings.* Tambov, TSU, 213 p. (2002)

[11] Akritas A.G., Malaschonok G.I. *Computation of Adjoint Matrix.* Computational Science, ICCS 2006, LNCS 3992, Springer, Berlin, 486-489.(2006)

[12] Malaschonok G. *On computation of kernel of operator acting in a module* Vestnik Tambovskogo universiteta. Ser. Estestvennye i tekhnicheskie nauki [Tambov University Reports. Series: Natural and Technical Sciences], vol. 13, issue 1,129-131 (2008)

[13] Malaschonok G. *Fast Generalized Bruhat Decomposition.* Computer Algebra in Scientific Computing, LNCS 6244, Springer, Berlin 2010. 194-202. DOI 10.1007/978-3-642-15274-0_16. arxiv:1702.07242 (2010)

[14] Malaschonok G. *On fast generalized Bruhat decomposition in the domains.* Tambov University Reports. Series: Natural and Technical Sciences. V. 17, Issue 2, P. 544-551. (http://parca.tsutmb.ru/src/MalaschonokGI17_2.pdf) (2012)

[15] Malaschonok G. *Generalized Bruhat decomposition in commutative domains.* Computer Algebra in Scientific Computing. CASC'2013. LNCS 8136, Springer, Heidelberg, 2013, 231-242. DOI 10.1007/978-3-319-02297-0_20. arxiv:1702.07248 (2013)

[16] Malaschonok G., Scherbinin A. *Triangular Decomposition of Matrices in a Domain.* Computer Algebra in Scientific Computing. LNCS 9301, Springer, Switzerland, 2015, 290-304. DOI 10.1007/978-3-319-24021-3_22. arxiv:1702.07243 (2015)

[17] Paul, Clayton R. *Fundamentals of Electric Circuit Analysis.* John Wiley & Sons. (2001). ISBN 0-471-37195-5.

[18] Rosenbrock, H.H. *Transformation of linear constant system equations.* Proc. I.E.E. V.114, 541544. (1967)

[19] Faugere, J.-C. *A new efficient algorithm for computing Gröbner bases (F4)* . Journal of Pure and Applied Algebra. Elsevier Science. Vol. 139, N.1, 61-88. (1999)

[20] Dumas, J.-G., Pernet, C., Sultan, Z. *Simultaneous computation of the row and column rank profiles.* In: Kauers, M. (Ed.), Proc. ISSAC13. ACM Press, pp. 181-188. (2013)

[21] Pernet C., Storjohann A. *Time and space efficient generators for quasiseparable matrices.* Journal of Symbolic Computation. V.85, N.2, 224-246. (2018)

[22] Ilchenko E.A. *An algorithm for the decentralized control of parallel computing process.* Tambov University Reports. Series: Natural and Technical Sciences, Vol. 18, No. 4, 1198-1206 (2013)

[23] Ilchenko E.A. *About effective methods parallelizing block recursive algorithms.* Tambov University Reports. Series: Natural and Technical Sciences, Vol. 20, No. 5, 1173-1186 (2015)