

Block Lanczos–Montgomery Method with Reduced Data Exchanges

Nikolai Zamarashkin and Dmitry Zheltkov^(✉)

INM RAS, Gubkina 8, Moscow, Russia
nikolai.zamarashkin@gmail.com, dmitry.zheltkov@gmail.com
<http://www.inm.ras.ru>

Abstract. We propose a new implementation of the block Lanczos–Montgomery method with reduced data exchanges for the solution of large linear systems over finite fields. The theoretical estimates obtained for parallel complexity indicate that the data exchanges in the proposed implementation for record-high matrix sizes and block sizes 50 require less time than those in ideally parallelizable computations with dense blocks. According to numerical results, the acceleration depends almost linearly on the number of cores (up to 2,000 cores). Then, the dependence becomes close to the square root of the number of cores.

Keywords: Large linear systems over finite field · Block Lanczos–Montgomery · Block Wiedemann–Coppersmith · Parallel computation

1 Introduction

The need to solve large linear systems over finite fields arises in a variety of applications, such as large composite number factorization and discrete logarithming in a large prime field. The most efficient methods for such problems are based on algorithms of two types:

1. Lanczos–Montgomery-like algorithms (see [3, 5, 6, 9, 10, 13, 14]);
2. Wiedemann–Coppersmith-like algorithms (see [1, 4, 7, 8]).

The algorithms of both types have much in common. Usually, the most time-consuming calculation is the repeated serial multiplication of a sparse matrix (and/or its transposed matrix) by a vector. The number of such multiplications is of the order of the doubled matrix size. Therefore, in terms of algorithmic complexity, both approaches are considered to be equivalent. However, when solving record-high-complexity problems, the efficient implementations on powerful computing systems with distributed memory plays a key role.

By now, the common opinion seems to be that the parallel versions of Lanczos–Montgomery-like techniques are generally inferior to Wiedemann–Coppersmith-like methods by their performance. All record-high RSA numbers were factorized using some modifications of the block Wiedemann–Coppersmith method for linear systems over \mathbb{F}_2 obtained from the GNFS method [7, 8].

The following simple idea of parallelization makes the Wiedemann-Coppersmith-like methods attractive. The idea is to multiply by a block consisting of K vectors, rather than use a multiplication of the matrix by a single vector. Since each column of the block can be multiplied independently, the algorithm acquires a considerable parallel resource.

Obviously, the similar block modification can be applied to the Lanczos–Montgomery-like algorithms (see, e.g., [14]). However, in this case, global data exchange occurs on each iteration. In the Montgomery method implemented in [14], the time required from this exchange was not scaled with the block-size growth, which led to a significant slowdown in the calculations. We believe that it is the problem of unscaled data exchanges that largely makes the Lanczos–Montgomery-like methods less popular for solving large sparse systems over finite fields.

In this paper, we propose an implementation of the block Lanczos–Montgomery method where the time required for global exchanges is scaled with the growth of the block size K . This implementation is an improvement of the one proposed in [14]. The theoretical estimates obtained for parallel complexity indicate that the data exchanges in the proposed implementation for record-high matrix sizes and block sizes of $K > 50$ require less time than those in ideally parallelizable computations with dense blocks.

It should be noted that the current practice of complexity analysis for methods of solving large sparse systems of linear equations over finite fields usually ignores the computing with dense blocks. It is assumed that the most complex part of the algorithm (the construction of the Krylov subspace basis) is associated with the multiplication of vectors by a large sparse matrix. However, as is shown below, this assumption is valid only for small blocks (with a size not exceeding 10).

The faster solution of the problem requires an increase in the block size of at least up to $K = 100$. In this case, the calculations with dense blocks cannot be disregarded. Now, the computation time even turns out to exceed the time required for exchanges. Thus, the more complex Wiedemann-Coppersmith-like algorithms actually have no significant advantages over the Lanczos-Montgomery methods; this is true at least for systems with a few dozens of thousands of distributed nodes (here, the number of computing cores can reach up to a million).

This study is organized as follows. Section 2 describes the algorithm and the method of data storage in the improved implementation of the Lanczos–Montgomery algorithm. Section 3 provides theoretical estimates for parallel complexity under the assumption that the computer system nodes are connected according to the “point-to-point” topology. This type of communication is currently widespread and can be implemented using the “fat-tree” topology. The numerical examples are presented in Sect. 4.

2 Description of the Improved Lanczos–Montgomery Algorithm

2.1 The Lanczos Algorithm for Linear Systems over Finite Fields

Like standard numerical methods for \mathbb{R} and \mathbb{C} fields, the Lanczos algorithm for the system $AX = B$ of linear equations over finite fields consists of the following steps:

1. transition to a symmetrized system of the form

$$A^T A X = A^T B, \quad (1)$$

2. calculation of the $A^T A$ -orthogonal basis V_0, V_1, \dots, V_N of the Krylov space constructed for the vectors $B, A^T A B, (A^T A)^2 B, \dots, (A^T A)^{\tilde{N}} B$, where \tilde{N} is an integer normally close to the number of rows N of A .

At the $(k+1)$ -th iteration, the new vector V_{k+1} is obtained from the short recurrence relations

$$V_{k+1} = A^T A V_k + \sum_{i=k}^{k-m} V_i C_{k+1,i}, \quad (2)$$

with a recursive depth m , where $m = 1$ for large finite fields and $m \geq 2$ for \mathbb{F}_2 .

3. iterative refinement of the solution

$$X_{k+1} = X_k + V_{k+1} G_{k+1}, \quad (3)$$

with some $K \times K$ matrix G_{k+1} .

However, implementation of the Lanczos algorithm over finite fields has some specific features. The first and most important feature is that the concept of “approximate solution” is meaningless in the case of finite fields. As a consequence, the number of vectors in the resulting $A^T A$ -orthogonal basis of the Krylov space is almost close (or equal) to the size of $A^T A$ (with a probability of 1). In addition, if the number of elements in \mathbb{F} is small, such as for \mathbb{F}_2 , block methods are used to avoid “breaks” [3, 5, 6, 14].

One of the most efficient implementations for \mathbb{F}_2 is the Montgomery implementation [5], where the block size is equal to the number of bits in the computer word. Finally, for small fields, to satisfy the condition of $A^T A$ -orthogonality, one has to increase the recursion depth up to $m \geq 2$.

In any case, the general structure of calculations in block Lanczos–Montgomery-like methods remains unchanged and can be described as follows:

The structure of Lanczos–Montgomery-like methods

1. **Start of iteration:** before the start of iteration i , the $N \times K$ blocks $V_i, V_{i-1}, \dots, V_{im}$ over the finite field \mathbb{F} are known;
2. **Matrix by block:** the product $(A^T A)V_i$ is calculated;
3. **Bilinear forms of blocks:** the bilinear forms $X^T Y \in \mathbb{F}^{K \times K}$ for the blocks $X, Y \in \mathbb{F}^{N \times K}$ are calculated;
4. **Linear combinations of blocks:** the linear combinations $XU + YV$ for the blocks $X \in \mathbb{F}^{N \times K}$ and square $K \times K$ matrices U, V are calculated.

2.2 Parallel Computing in the Improved Lanczos–Montgomery Method

The structure of the Lanczos method described in the previous section specifies the set of basic operations of the algorithm (see [13, 14]):

1. multiplication of a sparse matrix by block;
2. calculation of $X^T Y$ for $N \times K$ blocks X and Y ;
3. calculation of XU for $N \times K$ block X and $K \times K$ matrix U .

Our aim is to implement an efficient parallelization of these operations in the *improved Lanczos–Montgomery method*.

The improved Lanczos–Montgomery method proposed by us is based on unconventional ideas of the data storage on distributed nodes of the computer system and algorithms for computing with these data. This new way of data representation is very close to the one considered in [14] but there are some differences. These changes make it possible to obtain an implementation of the algorithm with a reduced number of data exchanges. In addition, as will be shown by further analysis and computational experiments, when the block size increases, the time required for exchanges becomes insignificant compared to the time of block operations. This occurs despite the fact that the arithmetic calculations are perfectly parallelized.

Now, we describe the method. Let K and S be two positive integers (hereafter, K denotes the block size). We consider a system of $K \times S$ nodes with distributed memory. For convenience, we assume that the nodes $\mathcal{N}_{i,j}$ form a rectangular lattice with K and S nodes along the rectangular grid sides. Despite the regular arrangement, we assume that the nodes are connected according to the “point-to-point” topology.

To fully describe the technique of data representation in memory, we consider:

1. the storage of a sparse matrix A ;
2. the storage of $N \times K$ blocks;
3. the storage of $K \times K$ dense matrices.

We begin with the large sparse matrix A . Despite the fact that a symmetrized system is considered in the algorithm, its matrix $A^T A$ is not calculated explicitly. To calculate $(A^T A)V$, we multiply the block V sequentially by the matrices A and A^T .

A special scheme is used to handle the matrix $A \in \mathbb{F}^{M \times N}$. We represent the matrix as a union of blocks of rows A^i and blocks of columns A_j :

$$A = \begin{bmatrix} A_1 & A_2 & \cdots & A_S \end{bmatrix} = \begin{bmatrix} A^1 \\ A^2 \\ \vdots \\ A^S \end{bmatrix}, \quad (4)$$

where each block of rows (columns) is assumed to have the same number of rows (columns) and approximately the same number of nonzero elements of A (this

can be achieved by using a special preprocessing technique). In the improved scheme, the node $\mathcal{N}_{i,j}$ contains both the block of rows A_j and the block of columns A^j .

Also, it is necessary to distribute the $N \times K$ blocks in the computer system memory. These blocks are stored in the following way: the block V is considered as the union of KS smaller blocks:

$$V = \begin{bmatrix} V_1 \\ V_2 \\ \dots \\ V_{KS} \end{bmatrix}, \quad (5)$$

and each subblock V_l of size $\frac{N}{KS} \times K$ is stored on the node $\mathcal{N}_{i,j}$ such that $l = K * j + i$.

Hereafter, the $K \times K$ matrices are supposed to be stored on each of the KS nodes $\mathcal{N}_{i,j}$.

This structure of data storage largely determines the parallel implementations for different operations in Lanczos–Montgomery-like method.

Remark 1. *The data storage technique considered above describes only the most general principles; some details of the efficient implementation were omitted. For example, some rows and/or columns of the sparse matrix A should be taken to be dense and stored without using special sparse formats. In the case of large fields, the dense rows (columns) can be of two types: (a) “small-module” elements; (b) “typical” elements for the large field. The sparse matrix is stored in a special cache-independent format, etc. However, to analyze the parallel complexity of the improved Lanczos–Montgomery method, it will suffice to use the rough description of data storage presented above.*

Now, we describe the algorithms. The multiplication of the symmetrized matrix of the system by a vector has the following form:

The algorithm of multiplication of the block X by the symmetrized matrix $A^T A$.

1. collect the vector X_i (the i -th column in the block X) on each node $\mathcal{N}_{i,j}$ for $j = 1, \dots, S$;
2. compute $Y_{i,j} = A^j X_i$ on the node $\mathcal{N}_{i,j}$;
3. collect the vector Y_i (the i -th column of $Y = AX$) on each node $\mathcal{N}_{i,j}$ for $j = 1, \dots, S$;
4. compute $W_{i,j} = A_j Y_i$ on the node $\mathcal{N}_{i,j}$ (the blocks of $W_{i,j}$ are the desired result);
5. collect $\frac{N}{KS} \times K$ blocks W_t , for $t = 1, \dots, KS$.

The data exchanges in **The algorithm of multiplication of the symmetrized matrix $A^T A$ by the block X** occur at steps 1, 3, and 5. The vectors are collected by calling the collective communication procedure **MPI_Allgather**. The amount of exchanged data does not exceed $2 \max(N, M)$.

The number of operations in this algorithm will be estimated in the next section for the case of systems over \mathbb{F}_2 .

For the block calculation, we propose

The algorithm for calculating bilinear forms X^TY .

1. compute $X_l^TY_l^T$ on the node $\mathcal{N}_{i,j}$, where $l = Kj + i$ (for all KS nodes);
2. collect the resulting matrix on all the nodes $\mathcal{N}_{i,j}$.

The data exchanges occur at step 2. The data are collected by calling the collective communication procedure **MPI_Allreduce**.

Finally, we consider the multiplication of the $N \times K$ block by the $K \times K$ matrix:

The algorithm of multiplication XU .

1. compute $W_l = X_lU$ on the node $\mathcal{N}_{i,j}$, where $l = Kj + i$ (for all KS nodes); the subblocks W_l are the computing result.

This algorithm has no data exchanges.

Remark 2. *The collective communication procedures such as **MPI_Allgather** and **MPI_Allreduce** are supposed to be effective if the nodes of the distributed system are connected according to the “point-to-point” topology.*

3 Parallel Complexity Analysis for the Improved Lanczos–Montgomery Method

3.1 Complexity Estimate for the Lanczos Method over Large Fields

Now, we estimate the parallel complexity of the Lanczos method for linear systems over large prime fields. The number of computer words for an element of the field is denoted by W , and the mean value of nonzeros in one column of A is p . For example, if the field element requires 512 bits, then $W = 8$. The parameter p may vary in a wide range (usually, its value is around 100). Our estimates are for the three main operations described in Subsect. 2.2.

Parallel complexity of the Lanczos method.

1. **The complexity of multiplication of a sparse matrix by a block:**

$$\{\text{computational complexity}\} = 2W \frac{pN^2}{KS}, \quad (6)$$

$$\{\text{complexity of data exchanges}\} = 2W \frac{N^2}{K}. \quad (7)$$

Note that the computational complexity (including the complexity of multiplications of numbers of a large field) depends only on the first degree of W (but not W^2 , as it could be expected). Estimate (6) holds for applications

where the initial matrix A is obtained by the GNFS method (for example, see [10]). In this case, almost all elements of A are “small-module” numbers (moreover, 90% of them are 1, and -1 in \mathbb{F}). As a result, the multiplication complexity is proportional to W .

Another important observation is that the time required for global exchanges is inversely proportional to the block size K . It is very important since K is the main parallelism resource in the algorithm. It should be noted that the term $2WpN^2$ in (6) is divisible by KS . It is this part of the algorithm that usually was most time-consuming. However, as it will be shown below, the situation changes significantly with the growth of K .

2. Computational complexity of bilinear forms:

$$\{\text{computational complexity}\} = 3 \frac{W^2 N^2}{S}, \quad (8)$$

$$\{\text{complexity of data exchanges}\} = WKN (\log_2 (KS) + 1). \quad (9)$$

The computational complexity of bilinear forms was estimated by us under the following assumptions: (a) the Montgomery algorithm was used for the multiplication of numbers in a large prime field and (b) the Winograd method was used for dense matrix multiplications. The Winograd algorithm allows us to reduce the number of multiplications of the elements of the large field in two times. It is the multiplications of elements that determine the complexity of algorithms of basic linear algebra in the case of large finite fields. (8) does not depend on K ; therefore, starting with some K (with a fixed S), the calculation time for block multiplications will surpass the time for the operations with the sparse matrix.

According to (9), the number of data exchanges grows with K ; however, the number of these exchanges in practice is rather small. Indeed, estimate (9) is linear over N . If the system size N is around several millions and K is almost 100, we have $KN \ll \frac{N^2}{K}$ and, therefore, the number of exchanges in (7) is much greater than that in (9).

3. Complexity of calculations of linear combinations of blocks:

$$\{\text{computational complexity}\} = \frac{9W^2 N^2}{2S}, \quad (10)$$

$$\{\text{complexity of data exchanges}\} = 0. \quad (11)$$

Like in (8), the complexity of computations in (10) corresponds to the Winograd method. The complexity value in (10) is independent of K . There are no data exchanges in this operation.

Remark 3.

It follows from the above estimates that the block operations are not scalable. Thus, the fast realizations of basic linear algebra procedures in finite fields are very important for the efficient implementation of the parallel Lanczos method. The more efficient are these calculations, the larger the block size can be taken.

3.2 Complexity Estimate for the Montgomery Method over \mathbb{F}_2

Let us estimate the complexity of parallel computations for the Montgomery method in the case of linear systems over \mathbb{F}_2 . The estimates are the same as in Subsect. 3.1 accurate to scalar coefficients.

Parallel complexity of the Montgomery method

1. Multiplication of sparse matrix by block:

$$\{\text{computational complexity}\} = \frac{1}{32} \cdot \frac{pN^2}{KS}, \quad (12)$$

$$\{\text{complexity of data exchanges}\} = \frac{N^2}{32K}. \quad (13)$$

The coefficient $\frac{1}{32}$ arises because the block size in the Montgomery method is divisible by the computer word size (which is supposed to be 64). The actual block size is taken to be $K \cdot 64$. Obviously, both terms are scaled with K .

2. Complexity of the calculation of bilinear forms:

$$\{\text{computational complexity}\} = 3 \frac{N^2}{8S}, \quad (14)$$

$$\{\text{complexity of data exchanges}\} = KN (\log_2(KS) + 1). \quad (15)$$

Estimate (14) was obtained for the case of the Coppersmith algorithm, which allows one to reduce the number of operations with computer words in 8 times compared to the “naive algorithm”.

3. Complexity of the linear combination of blocks:

$$\{\text{computational complexity}\} = \frac{5N^2}{8S}, \quad (16)$$

$$\{\text{complexity of data exchanges}\} = 0. \quad (17)$$

Here, the “four Russians” method is used for the dense matrix multiplications in \mathbb{F}_2 . This algorithm makes it possible to reduce the number operations with computer words in 8 times compared to the “naive” algorithm.

3.3 Parallel Complexity Analysis for the Lanczos–Montgomery-like Methods

We use the estimates obtained in Sects. 3.1 and 3.2 to demonstrate that the main obstacle to the use of more powerful computer systems for Lanczos–Montgomery-like methods is the operations with dense matrices and blocks, rather than data exchanges.

First, we determine the values of parameters K and S for which the complexity of multiplications of sparse matrix by blocks and the complexity of calculations of bilinear forms coincide. This can be easily done by equaling the expressions (6) and (8). In this case, we have

$$\frac{p}{K} = 3W. \quad (18)$$

For $p = 200$, and $W = 8$, we have $K \leq 9$. In other words, even for K of around 10 the parallel complexity (8) is no less than (6). This means that the parallel resource of K is bounded.

Remark 4.

In fact, the situation is slightly better: sparse matrices usually have a random structure; therefore, the speed of operations with them is limited due to inefficient memory access. Therefore, it will be more correct to use an estimate for values of K close to 100. In addition, one can significantly speed up the linear algebra operations (BLAS) by using special accelerators. For example, in case of basic algebra operations over large fields, GPU can be used. The resulting acceleration can increase the block size K . Also, the use of fast algorithms leads to a further acceleration with the increase of K . In this case, estimate (8) should be refined.

Now, we compare the times required for exchanges and computations with dense blocks. We assume the following distributed computing system:

1. the communication network of the computer system efficiently links the nodes via the “point-to-point” topology (for example, using the “fat-tree” technology);
2. the baud rate is about 20 Gb/s (approximately $300 \cdot 10^8 \frac{W}{c}$);
3. we assume that the computer system is equipped with 8-core nodes running at $3 \cdot 10^8$ GHz. Thus, each node is capable of performing $2.4 \cdot 10^{10} \frac{W}{c}$ computer words operations per second.

The most important consequence of the above description is that the calculations performed by a single multicore node are 100 times faster than the its data transmissions.

By equating (7) and (8), we find the values of K and S such that the times for data exchanges and for computations with dense blocks are comparable:

$$\frac{K}{S} = 100 \frac{2}{3W}. \quad (19)$$

Taking into account that $S \geq 10$ (usually), we obtain $K \geq 100$. Thus, the data exchanges in the improved algorithm imposes less strict limitations on K than the computations with dense blocks.

Similarly, we can show that the same conclusions hold for the field \mathbb{F}_2 . Note that these conclusions are independent of the matrix sizes. However, they depend on the average number of nonzeros in row p . The larger is p , the larger values of K can be used and the higher is the parallel efficiency of the improved method.

4 Numerical Experiments

The following tables give the absolute times obtained in numerical experiments for the improved Lanczos–Montgomery method applied to linear systems over finite fields. The numbers of system cores are indicated in the tables. All the

Table 1. Acceleration and the time for single basis vector calculation in the case of a large finite field. Block Lanczos method (matrix $2 \cdot 10^6$, with 84 nonzero elements in a column)

Number of cores	1	2	4	8	16	32	64	128
Acceleration	1	1.99	3.95	7.86	15.48	30.49	56.04	103.04
Time, s	38.1	19.2	9.6	4.85	2.46	1.25	0.67	0.37
$S \times K$	1×1	1×1	1×1	1×1	2×1	2×2	4×2	8×4

experiments were performed on the Lomonosov and Lomonosov-2 supercomputers. In the experiments, the matrices were equivalent to the matrices obtained by the GNFS method. In the case of large fields, the numbers were encoded using 512 bits. It should be noted that the block size in some experiments was sufficiently large: $K = 32$. According to the Tables 1, 2, 3 and 4, the resulting acceleration depends almost linearly on the number of cores (up to 2,000 cores). Then, the dependence becomes close to the square root of the number of cores.

Table 2. Acceleration and the time for single basis vector calculation in the case of a large finite field. Block Lanczos method (matrix $2 \cdot 10^6$, with 84 nonzero elements in a column)

Number of cores	256	512	1024	2048	4096	8192
Acceleration	159.48	231.01	391.34	608.96	847.13	1211.4
Time, s	0.239	0.165	0.0974	0.0625	0.0451	0.0316
$S \times K$	16×4	16×8	32×8	32×16	64×16	64×32

Table 5 contains the absolute times of computations and data exchanges for different values of S and K in case of the field \mathbb{F}_2 . The computations include:

- Sparse matrix by vector product (‘MatVec’);
- All operations with the dense blocks (‘Dense’).

The data exchanges occur in the following operations:

- Matrix by vector multiplication (‘MatVec’);
- Collecting vectors and subblocks before and after matrix by vector multiplication (‘Block-Vector’);
- Bilinear forms for dense blocks (‘Dense’).

Synchronizations constitute a substantial part of the time, especially for the exchanges in ‘Block-Vector’ and ‘Dense’ parts. That is why the numerical results confirm the theoretical estimates imperfectly for the data exchanges times. However, the results are close to the theory.

Table 3. Acceleration and the time for single basis vector calculation in the case of a field \mathbb{F}_2 . Block Lanczos method (matrix $2 \cdot 10^6$, with 300 nonzero elements in a row)

Number of cores	1	2	4	8	16	32	64	128	256	512	1024
Acceleration	1	1.88	3.6	6.8	12.4	23.8	43.9	82.4	153.54	268.2	342.0
Time, ms	57.1	30.4	15.9	8.39	4.59	2.40	1.30	0.693	0.372	0.213	0.167
$S \times K$	1×1	1×1	1×1	1×1	2×1	4×1	4×2	8×2	8×4	16×4	16×8

Table 4. Acceleration and the time for single basis vector calculation in case of a field \mathbb{F}_2 . Block Lanczos method (matrix $2 \cdot 10^7$, with 800 nonzero elements in a row)

Number of cores	1024	2048	4096	8192
Acceleration	342.0	605.1	813.7	1140.2
Time, ms	1.38	0.780	0.579	0.414
$S \times K$	16×8	32×8	64×16	64×32

Table 5. Computation and communication time of different operations (\mathbb{F}_2 field, matrix 2127498×2127690 , 170 nonzero entries per row)

Parameters			Computation time, s		Communication time, s		
Number of cores	S	K	MatVec	Dense	MatVec	Block-Vector	Dense
14	1	1	2396	382	0	0	0
28	2	1	1220	186	193	0	6
28	1	2	1223	382	0	130	1.5
56	4	1	553	105	206	0	18
56	2	2	692	194	102	70	18
56	1	4	690	380	0	114	8
112	8	1	280	49	200	0	15
112	4	2	277	102	142	39	19
112	2	4	351	195	62	56	19
224	16	1	149	28	244	0	6
224	8	2	142	55	148	21	12
224	4	4	137	105	88	41	14

Acknowledgments. The work was supported by the Ministry of Education and Science of the Russian Federation, agreement no. 14.604.21.0034 (identifier RFMEFI60414X0034).

References

1. Wiedemann, D.H.: Solving sparse linear equations over finite fields. IEEE Trans. Inform. Theor. **32**(1), 54–62 (1986)

2. Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stan.* **45**, 255–282 (1950)
3. Coppersmith, D.: Solving linear equations over $\text{GF}(2)$: block Lanczos algorithm. *Linear Algebra Appl.* **193**, 33–60 (1993)
4. Coppersmith, D.: Solving homogeneous linear equations over $\text{GF}(2)$ via block Wiedemann algorithm. *Math. Comput.* **62**(205), 333–350 (1994)
5. Montgomery, P.L.: A block Lanczos algorithm for finding dependencies over $\text{GF}(2)$. In: Guillou, L.C., Quisquater, J.-J. (eds.) *EUROCRYPT 1995*. LNCS, vol. 921, pp. 106–120. Springer, Heidelberg (1995). doi:[10.1007/3-540-49264-X_9](https://doi.org/10.1007/3-540-49264-X_9)
6. Peterson, M., Monico, C.: \mathbb{F}_2 Lanczos revisited. *Linear Algebra Appl.* **428**, 1135–1150 (2008)
7. Kleinjung, T., et al.: Factorization of a 768-Bit RSA modulus. In: Rabin, T. (ed.) *CRYPTO 2010*. LNCS, vol. 6223, pp. 333–350. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-14623-7_18](https://doi.org/10.1007/978-3-642-14623-7_18)
8. Thome, E., et al.: Factorization of RSA-704 with CADO-NFS, pp. 1–4. Preprint (2012)
9. Dorofeev A.Y.: Vychislenie logarifmov v konechnom prostom pole metodom lineinogo resheta. (Computation of logarithms over finite prime fields using number sieving) *Trudy po diskretnoi matematike*, vol. 5, pp. 29–50 (2002)
10. Dorofeev, A.Y.: Solving systems of linear equations arising in the computation of logarithms in a finite prime field. *Math. Aspects Crypt.* **3**(1), 5–51 (2012). (In Russian)
11. Cherepnev, M.A.: A block algorithm of Lanczos type for solving sparse systems of linear equations. *Discret. Math. Appl.* **18**(1), 79–84 (2008)
12. Cherepnev, M.A.: Version of block Lanczos-type algorithm for solving sparse linear systems. *Bull. Math. Soc. Sci. Math. Roumanie* **53**, 225–230 (2010). Article no. 3. <http://rms.unibuc.ro/bulletin>
13. Popovyan, I.A., Nestrenko, Y.V., Grechnikov, E.A.: Vychislitelno slozhnye zadachi teorii chisel. Uchebnoe posobie (Computationally hard problems of number theory. Study guide). Publishing of the Lomonosov Moscow State University (2012)
14. Zamarashkin, N.L.: Algoritmy dlya razrezhennykh sistem lineinykh uravneniy v $\text{GF}(2)$. Uchebnoe posobie (Algorithms for systems of linear equations over $\text{GF}(2)$. Study guide). Publishing of the Lomonosov Moscow State University (2013)

Supercomputing

Second Russian Supercomputing Days, RuSCDays

2016, Moscow, Russia, September 26–27, 2016,

Revised Selected Papers

Voevodin, V.; Sobolev, S. (Eds.)

2016, XV, 370 p. 166 illus., Softcover

ISBN: 978-3-319-55668-0