Preface

The interest in parallel computer algebra and parallel symbolic computations appeared more than 20 years ago. Earlier meetings that concerned the Parallel Symbolic Computation were held in Linz, Austria, (PASCO'94) and in Maui, U.S.A., (PASCO'97).

After 10 years gap the interest in this field has started to grow for the last 5 years. There were two PASCO conferences at London, Canada (PASCO'07) and in Grenoble, France (PASCO'10) and four international conferences "Applications of Computer Algebra" where special sessions of parallel computations were organized. There were special sessions "Parallel Computer Algebra" at ACA'2006 in Varna, Bulgaria and at ACA'2008 in Linz, Austria, organized by Gennadi Malaschonok, Tambov University, Russia, the session "High-Performance Computer Algebra" organized by Jeremy Johnson, Drexel University, USA, and Marc Moreno Maza, University of Western Ontario, Canada, at ACA'2009 in Western Ontario, Canada and the session "Parallel Computations" organized by Gennadi Malaschonok and Stephen Watt, Western Ontario, Canada, at ACA'2010 in Vlora, Albania.

Therefore, the organization of a new international conference "Parallel Computer Algebra" (ParCA-2010) in Tambov University (Russia) is the natural consequence of the growing interest shown by the mathematicians in this field of computer science and particular in the growing activity of Russian researchers in this direction.

This volume of the Tambov University Report contains revised version of the papers submitted to the international conference "Parallel Computer Algebra" (ParCA'2010) by the participants and accepted by the program committee after a through reviewing process. The general areas of interest of ParCA 2010 conference include all aspects of parallel algorithms for computer algebra, software techniques for parallel computer algebra systems, applications of parallel computer algebra in all fields and using computer algebra in designing parallel algorithms or software in other areas.

The topics include:

– parallel polynomial computation,
– parallel algorithms for symbolic linear algebra, matrix operations and linear systems,
– parallel methods for solving systems of differential equations,
– parallel methods for Groebner basis computation,
– parallel algorithms in combinatorics and cryptography,
– parallel algorithms in computational algebraic geometry,
– complexity of parallel computer algebra algorithms,
– reinvention and adaptation of existing symbolic algorithms to a parallel setting.

A total of 18 contributions were received in response to the call for papers. These were reviewed by members of the Program Committee or by external reviewers selected by the committee. Each paper received between two and four reviews, with most receiving three. Finally, 11 papers (about 60 inclusion in these proceedings). While the majority of the authors are from the Russian Federation, it is a pleasure to see a true international flavor at this meeting, with authors and PC members representing eleven countries on three continents.

The "Parallel Computer Algebra" conference was supported financially by the Russian Foundation of Basic Research grant 10-01-06045g, by the Administration of Tambov Region and by the Tambov State University. We are grateful for the support. Our special gratitude to the members of ParCA Program Committee and to the members of the ParCA Local Organizing Committee in Tambov.

June 2010 Gennadi Malaschonok

Organization

PARCA 2010 was organized by the Institute of Mathematics, Physics and Informatics of Tambov State University, Tambov, Russia.

Organizing Committee:

General Chair and PC Co-Chair: Gennadi Malaschonok, Tambov State U., Russia
PC Co-Chair: Gene Cooperman, Northeastern U., Boston, USA
PC Co-Chair: Stephen M. Watt, U. Western Ontario, Canada
Local Arrangements Chair: Natalia Malaschonok, Tambov State U., Russia

Committee:

Yuri Blinkov, Saratov State U., Russia,
Gene Cooperman, Northeastern U., Boston, USA,
James H. Davenport, U. Bath, UK,
Jean-Guillaume Dumas, Université Joseph Fourier, France,
Vladimir Gerdt, Dubna, JINR, Russia,
Jeremy Johnson, Drexel U., USA,
Erich Kaltofen, North Carolina State U., USA,
Tony Kennedy, U. Edinburgh, UK,
Viktor Levandovskyy, Aachen U., Germany,
Gennadi Malaschonok, Tambov State U., Russia,
Marc Moreno Maza, U. Western Ontario, Canada,
Aleksandr Myllari, Turku U., Finland,
Dana Petcu, Western U. of Timisoara, Romania,
Alexander Tiskin, Warwick U., UK,
Nikolay Vasiliev, PDMI RAS, St. Petersburg, Russia,
Stephen M. Watt, U. Western Ontario, Canada,
Alexey Zobnin, Moscow State U., Russia.

Local Arrangements Committee:

Andrey Betin,
Alexey Lapaev (web-master),
Natalia Malaschonok,
Oxana Pereslavtseva (secretary),
Maxim Starov.

UDC 519.612

ESTIMATES OF THE RUNNING TIME AND MEMORY REQUIREMENTS OF THE NEW ALGORITHM OF SOLVING LARGE SPARSE LINEAR SYSTEMS OVER THE FIELD WITH TWO ELEMENTS

© **Vasiliy Vadimovich Astakhov**
Moscow State University named after M.V. Lomonosov, Leninskie Gory, 1, Moscow, 119991, Russia, Numbers Theory Department of Information Protection Branch, Student of Mechanics and Mathematics Faculty, e-mail: astvvas89@mail.ru

Key words: linear sparse systems; pade approximations; corank distribution.
A new algorithm of solving large sparse linear systems over field with two elements is considered in this work. Algorithm was proposed by M.A. Cherepniov. Algorithm uses the construction of matrix Pade approximations over finite fields. It is supposed that elements of approximation polynomials are independent and are identically distributed. Method for finding distributions of coranks of random symmetric, antisymmetric and common matrices is constructed. Lower and upper bounds for number of previous approximations sufficient to construct the next one are obtained. The logarithmic dependence for sufficient number of keeping approximations on every step for successful completion of algorithm with probability of 0.99 is found. Using the computer program exact values of estimates of running time and memory requirements are found, results are given in this work.

1 Introduction

A new algorithm of solving sparse linear systems over Z_2 is considered in [1]. Algorithm uses the construction of matrix Pade approximations over Z_2. Let

$$\alpha = \Sigma_{i=0}^{\infty}\alpha_i\lambda^{-i}, \alpha_i \in F(n\times n), n \in N, n \geqslant 64,$$

where λ is transcendental variable.

Matrix polynomials $Q^{(s)}(\lambda) \in F(n\times n)[\lambda]$ that satisfy

$$\alpha(\lambda)Q^{(s)}(\lambda) - P^{(s)}(\lambda) = \Sigma_{i=s+1}^{\infty}p_{i,s}\lambda^{-i}$$

$$degQ^{(s)} \leqslant s, degP^{(s)} \leqslant s,$$

for some $P^{(s)}(\lambda) \in F(n\times n)[\lambda]$, are said to be matrix Pade approximations with number s, or simply s-approximation of the series $\alpha(\lambda)$.

Denote the coefficients of approximation according to the formula:

$$Q^{(s)}(\alpha) = \Sigma_{i=0}^{s}Q_i^{(s)}\cdot\lambda^i.$$

Let $Q_s^{(s)}$ vanishes. Consider following transformation. With the help of elementary transformations of the columns of matrix polynomial $Q^{(s)}(\lambda)$ we reduce its leading term to the form where left columns are linearly independent and the rest are zeros. Then we multiply columns of reduced

matrix polynomial $Q(s)(\lambda)$ corresponding to the zero columns of its leading term on λ. Repeat such procedures until the leading term become nonsingular. Denote the necessary number of such iterations by ζ_s. There was shown in [1], that for $s > k-1$ fulfillment of conditions:

$$\zeta_s \leqslant k-1, \zeta_{s-i} \leqslant k-i, i = 1, ..., k-1,$$

is sufficient for constructing approximation with the number $s+1$, with the help of coefficients of approximations with numbers $s-k+1, ..., s$.

In this paper we obtain estimates on the expectation of the running time and memory requirements for the successful completion of the algorithm with high probability.

In the second part of the work we obtain the distribution of coranks of random square matrices over finite fields. In the third part estimates on the distribution of stohastic variable ζ are obtained. In the fourth part we make the estimates on the expected value and the distribution of auxiliary stochastic variable τ. In the fifth and sixth parts we obtain final results and give some experimentally established facts.

2 Distribution of coranks of random matrices over Z_p

Consider a matrix, whose elements are independent identically distributed random variables with values in Z_p, where p is a prime number. Denote $q = \frac{1}{p}$.

$a_{nm}[r]$ - probability that the matrix of size $n \times m$ with values in Z_p has rank r. $t_n[r]$ - probability that the matrix of size $n \times n$ with values in Z_p has corank r. Let $b_n[r] = a_{nn}[r]$. $t_n[r] = b_n[n-r]$

$B_n(x)$ - generating function for the distribution of ranks of matrices of size $n \times n$. $T_n(x)$ - generating function for the distribution of coranks of matrices of size $n \times n$. $B_n(x) = x^n \cdot T_n(\frac{1}{x})$

$s_n[r]$ - probability that the symmetric matrix of size $n \times n$ with values in Z_p have rank r. $f_n[r]$ - probability that the symmetric matrix of size $n \times n$ with values in Z_p have corank r. $s_n[r] = f_n[n-r]$

$S_n(x)$ - generating function for the distribution of ranks of symmetric matrices of size $n \times n$. $F_n(x)$ - generating function for the distribution of coranks of symmetric matrices of size $n \times n$.

$S_n(x) = x^n \cdot F_n(\frac{1}{x})$

$P(A)$ - probability of the event A.

Statement 1

$$a_{nm}[r] = a_{(n-1)m}[r] \cdot p^{r-m} + a_{(n-1)m}[r-1] \cdot (1 - p^{r-m-1}) \quad n, m, r > 0 \tag{1}$$

Proof 1 *If the matrix E' size $n \times m$ has rank r, then its submatrix without the last row (of size $n-1 \times m$) may have a rank r or $r-1$. In the first case n-th row is in the linear span of the previous rows with probability $\frac{p^r}{p^m}$. In the second it is not in the linear span with probability $1 - \frac{p^{r-1}}{p^m}$. We obtain the required equality.* $\square$

Now consider the method that will help us not only to solve this problem for square matrices, but also for symmetric matrices.

Consider the matrix E of size $n \times n$, $rankE = r$ and discard its last column and row. The resulting matrix may have a rank r, $r-1$ or $r-2$. Denote the last row without the last

element for $\bar{\alpha}^t$, the last column (without the last element) for $\bar{\beta}$, an angular element for e, and the matrix $(n-1) \times (n-1)$ for E'.

Lemma 1 *Let the matrix E' has rank j.*

1. *$\bar{\beta}$ is not in the linear span of its columns, and $\bar{\alpha}^t$ is not in the linear span of its rows. Then, regardless of the value of e matrix E has rank $j+2$.*
2. *$\bar{\beta}$ is in the linear span of its columns, but $\bar{\alpha}^t$ is not in the linear span of its rows. Then, regardless of the value of e matrix E has rank $j+1$.*
3. *$\bar{\beta}$ is in the linear span of its columns, and $\bar{\alpha}^t$ is in the linear span of its rows. Then for exactly one value of the element e matrix E has rank j, and for the rest of them - rank $j+1$.*

Proof 2

1. *Consider the first $n-1$ rows of the matrix E, due to the fact, that $\bar{\beta}$ is not in the linear span of the columns of E', rank of such submatrix is equal to $rankE'+1 = j+1$. $\bar{\alpha}^t$ is not in the linear span of the rows of E', hence the last row of E is not in the linear span of the first $n-1$ row. Therefore rank of is equal to $j+2$.*
2. *Consider the first $n-1$ rows of the matrix E, due to the fact, that $\bar{\beta}$ is in the linear span of the columns of E', rank of such submatrix is equal to $rankE' = j$. $\bar{\alpha}^t$ is not in the linear span of the rows of E', hence the last row of E is not in the linear span of the first $n-1$ rows. Therefore rank of is equal to $j+1$.*
3. *Consider the first $n-1$ rows of the matrix E, due to the fact, that $\bar{\beta}$ is in the linear span of the columns of E', rank of such submatrix is equal to $rankE' = j$. $\bar{\alpha}^t$ is in the linear span of the rows of E', therefore it is a linear combination of these rows. If we consider the linear combination of the rows of E with the same coefficients, we get a row with the first $n-1$ elements identical to $\bar{\alpha}^t$, and the last one is equal to some value e_1. If $e = e_1$ then last row is in the linear span of the first $n-1$ and we obtain $rankE = j$. Now we will show, that if e is not equal to e_1 rank of E is $j+1$. Assume the oppose, $rankE = j$ and the last row is a linear combination of the first $n-1$. Then consider the row $H = (0, 0...0, e-e_1)$, it is also a linear combination of the first $n-1$ rows of E, hence the rows of matrix E with 0 on the last position are linear combinations of the first $n-1$ rows of E. Considering these rows and H we get that the submatrix of first $n-1$ rows of E have rank $rankE'+1$, but it has to be equal to $j = rankE'$. Contradiction ends the proof.* $\square$

Theorem 1

$$b_n[r] = b_{n-1}[r] \cdot p^{2\cdot(r-n)+1} + b_{n-1}[r-1] \cdot (p^{2\cdot(r-n)-1} \cdot (p-1) + 2 \cdot p^{r-n} \cdot (1-p^{r-n})) +$$

$$+ b_{n-1}[r-2] \cdot (1-p^{r-n-1}) \cdot (1-p^{r-n-1}), \quad n, r > 0. \tag{2}$$

Proof 3 *With the help of Lemma 1 we obtain:*

Probability, that E *has rank* r, *under condition, that* E' *has rank* $r-2$, *is equal to* $p_1 = (1 - \frac{p^{r-2}}{p^{n-1}})^2$, *because it is possible, if and only if the statements of the first item are fulfilled.*

Probability, that E *has rank* r, *under condition, that* E' *has rank* $r-1$, *is equal to* $p_2 = 2 \cdot (1 - \frac{p^{r-1}}{p^{n-1}}) \cdot \frac{p^{r-1}}{p^{n-1}} + (\frac{p^{r-1}}{p^{n-1}})^2 \cdot \frac{p-1}{p}$, *because it is possible, if and only if statements of the second item are fulfilled or statements of the third item are fulfilled and* $e \neq e_1$.

Probability, that E *has rank* r, *under condition, that* E' *has rank* r, *is equal to* $p_3 = (\frac{p^r}{p^{n-1}})^2 \cdot \frac{1}{p}$, *because it is possible, if and only if statements of the third item are fulfilled and* $e = e_1$.

We have:

$$b_n[r] = b_{n-1}[r-2] \cdot p_1 + b_{n-1}[r-1] \cdot p_2 + b_{n-1}[r] \cdot p_3.$$

Substituting p_1, p_2, p_3 *with found expressions we obtain the required equality.* □

Corollary 1

$$B_n[x] = B_{n-1}(x \cdot p^2) \cdot p^{1-2 \cdot n} + x \cdot B_{n-1}(x \cdot p^2) \cdot (p^{1-2 \cdot n} \cdot (p-1) - 2 \cdot p^{2-2 \cdot n}) + x \cdot B_{n-1}(x \cdot p) \cdot 2 \cdot p^{1-n} +$$

$$+ x^2 \cdot (B_{n-1}(x) - 2 \cdot B_{n-1}(x \cdot p) \cdot p^{1-n} + B_{n-1}(x \cdot p^2) \cdot p^{2-2 \cdot n}) =$$

$$= p^{1-2 \cdot n} \cdot (x \cdot p - 1) \cdot (x-1) B_{n-1}(x \cdot p^2) + 2 \cdot p^{1-n} \cdot x \cdot (1-x) \cdot B_{n-1}(x \cdot p) + x^2 \cdot B_{n-1}(x) \quad (3)$$

Proof 4 *Consider the coefficient of* x^r *in the right hand side.* $b_{n-1}[r] \cdot p^{2 \cdot r} \cdot p^{1-2 \cdot n}$ *in the first summand,* $b_{n-1}[r-1] \cdot p^{2 \cdot r-2} \cdot (p^{1-2 \cdot n} \cdot (p-1) - 2 \cdot p^{2-2 \cdot n})$ *- in the second,* $b_{n-1}[r-1] \cdot p^{r-1} \cdot 2 \cdot p^{1-n}$ *- in the third,* $b_{n-1}[r-2] \cdot (1 - 2 \cdot p^{r-2} \cdot p^{1-n} + p^{2 \cdot r-4} \cdot p^{2-2 \cdot n}$ *- in the fourth. It is identical to the expression in (2) for* $b_n[r]$. □

Corollary 2

$$s_n[r] = s_{n-1}[r] \cdot p^{r-n} + s_{n-1}[r-1] \cdot p^{r-n-1} \cdot (p-1) + s_{n-1}[r-2] \cdot (1 - p^{r-1-n}) \quad (4)$$

Proof 5 *Proof is similar to the Theorem 1, but we have to consider the fact, that some of the conditions of Lemma 1 can't be satisfied. Probability, that* E *has rank* r, *under condition, that* E' *has rank* $r-2$ *is equal to* $p_1 = (1 - p^{r-1-n})$, *because it is possible, if and only if the statement of the first item is fulfilled.(If the last row is not in the linear span of first* $n-1$ *columns (without last elements), then due to the simmetry the last column is not in the linear span of first* $n-1$ *rows (without last elements))*

Probability, that E *has rank* r, *under condition, that* E' *has rank* $r-1$ *is equal to* $p_2 = p^{r-n-1} \cdot (p-1)$, *because it is possible, if and only if the statement of the third item is fulfilled and* $e \neq e_1$.

Probability, that E *has rank* r, *under condition, that* E' *has rank* r *is equal to* $p_3 = p^{r-n}$, *because it is possible, if and only if the statement of the third item is fulfilled and* $e = e_1$. *We have:*

$$s_n[r] = s_{n-1}[r-2] \cdot p_1 + s_{n-1}[r-1] \cdot p_2 + s_{n-1}[r] \cdot p_3.$$

Substituting p_1, p_2, p_3 *with found expressions we obtain the required equality.* □

Corollary 3

$$S_n(x) = (q^n + x \cdot (1-q) \cdot q^{n-1} - q^{n-1} \cdot x^2) S_{n-1}(x \cdot p) + x^2 \cdot S_{n-1}(x) =$$

$$= q^{n-1} \cdot (1-x) \cdot (q+x) \cdot S_{n-1}(x \cdot p) + x^2 \cdot S_{n-1}(x). \quad (5)$$

Proof 6 *Proof is similar to the proof of the Corollary 1. Consider the coefficient of* x^r *in the left and right hand sides. In the left it is equal to* $s_n[r]$, *and in the right to* $q^{n-r} \cdot s_{n-1}[r] + (1-q) \cdot q^{n-r} \cdot s_{n-1}[r-1] + (1-q^{n-r+1}) \cdot s_{n-1} r - 2$. *Required equality is the result of Corollary 2.* □

Due to the equalities $B_n(x) = x^n \cdot T_n(1/x)$ и $S_n(x) = x^n \cdot F_n(1/x)$, we obtain

$$x \cdot T_n(x) = (q \cdot x - 1) \cdot (x-1) \cdot T_{n-1}(x \cdot q^2) + 2 \cdot (x-1) \cdot T_{n-1}(x \cdot q) + T_{n-1}(x), \quad (6)$$

$$x \cdot F_n(x) = (x-1) \cdot (q \cdot x + 1) \cdot F_{n-1}(x \cdot q) + F_{n-1}(x). \quad (7)$$

Define formal power series T, F from the following equations:

$$T(x) = (q \cdot x - 1) \cdot T(x \cdot q^2) + 2 \cdot T(x \cdot q), \quad (8)$$

$$F(x) = (q \cdot x + 1) \cdot F(x \cdot q). \quad (9)$$

They are obtained from(8), (9) by replacement of T_n, T_{n-1} and F_n, F_{n-1} by T and F respectively.

Denote $f[r]$ and $t[r]$ as coefficients of x^r in $F(x)$ and $T(x)$, respectively. Obtain them, considering coefficients of x^r in (8), (9):

$$t[r] = t[r-1] \cdot \frac{q^{2 \cdot r - 1}}{(1-q^r)^2}, \quad (10)$$

$$f[r] = f[r-1] \cdot \frac{q^r}{1-q^r} = \frac{f[r-1]}{p^r - 1}. \quad (11)$$

Whence we obtain the expressions:

$$T(x) = t_q \cdot (1 + \frac{q \cdot x}{(1-q)^2} + \frac{q^4 \cdot x^2}{((1-q)(1-q^2))^2} + \ldots + \frac{q^{k^2} \cdot x^k}{((1-q)(1-q^2)\ldots(1-q^k))^2} + \ldots), \quad (12)$$

where we set $t_q = (1-q)(1-q^2)\ldots(1-q^k)\ldots$.

$$F(x) = c_q \cdot (1 + \frac{x}{p-1} + \frac{x^2}{(p-1)(p^2-1)} + \ldots + \frac{x^k}{(p-1)(p^2-1)\ldots(p^k-1)} + \ldots), \quad (13)$$

where we set c_q, such that $F(1) = 1$. To find c_q we prove auxillary Lemma.

Lemma 2

$$\Sigma_{i=0}^{\infty} \Pi_{j=1}^{i} \frac{1}{p^j - 1} = \Pi_{i=0}^{\infty} \frac{1}{1 - q^{2 \cdot i + 1}} \quad (14)$$

Proof 7 *Let prove, that*

$$\Sigma_{i=0}^{2\cdot k+1}\frac{1}{\Pi_{j=1}^{i}(p^j-1)}=\Sigma_{i=1}^{k+1}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)} \tag{15}$$

by the mathematical induction. Base case for $k=0,\quad 1+\frac{1}{p-1}=\frac{p}{p-1}$. *To prove the inductive step note the following equality:*

$$p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}+p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}=$$

$$=\frac{p^{i^2+2\cdot k+3}}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}=$$

$$=p^{(i+1)^2}\frac{1}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i}(p^{2\cdot j}-1)}+p^{(i+1)^2}\frac{1}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}. \tag{16}$$

Now with the help of induction hypothesis:

$$\Sigma_{i=0}^{2\cdot k+3}\frac{1}{\Pi_{j=1}^{i}(p^j-1)}=\Sigma_{i=1}^{k+1}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}+$$

$$+\frac{1}{\Pi_{j=1}^{2\cdot k+2}(p^j-1)}+\frac{1}{\Pi_{j=1}^{2\cdot k+3}(p^j-1)}=$$

From (16) with $i=0$ *we obtain:*

$$=\Sigma_{i=1}^{k+1}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}+\frac{p}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k}(p^{2\cdot j}-1)}+$$

$$+\frac{p}{\Pi_{j=1}^{2\cdot k+3}(p^j-1)}=\Sigma_{i=2}^{k+1}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}+$$

$$+\frac{p^4}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-1}(p^{2\cdot j}-1)}+\Sigma_{i=1}^{2}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+2}(p^{2\cdot j}-1)}$$

With the help of (16) with $i=2,\ldots,l-1$ *we get:*

$$=\ldots=\Sigma_{i=l}^{k+1}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}+$$

$$+\frac{p^{l^2}}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k+1-l}(p^{2\cdot j}-1)}+\Sigma_{i=1}^{l}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+2}(p^{2\cdot j}-1)}=$$

And in the result:

$$=\ldots=\Sigma_{i=1}^{k+2}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k+1}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+2}(p^{2\cdot j}-1)}.$$

In the left side of (15) are partial sums of the series from (14). And in the right:

$$\Sigma_{i=1}^{k+1}p^{i^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}\cdot\frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)}\geqslant p^{(k+1)^2}\cdot\frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)}.$$

$$\Sigma_{i=1}^{k+1} p^{i^2} \cdot \frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)} \cdot \frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)} =$$

$$= p^{(k+1)^2} \cdot \frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)} + p^{(k+1)^2} \cdot \frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)} \cdot \Sigma_{i=1}^{k} p^{(k+1)^2-i^2} \cdot \frac{1}{\Pi_{j=1}^{k-i+1}(p^{2\cdot j}-1)} \leqslant$$

$$\leqslant p^{(k+1)^2} \cdot \frac{1}{\Pi_{j=0}^{k}(p^{2\cdot j+1}-1)} \cdot (1 + k \cdot p^{-(2\cdot k+1)}).$$

Whence we obtain, that expression in the right side of (15) tends to the right side of (14). Lemma is proved. □

Therefore $c_q = (1-q)(1-q^3)\dots(1-q^{2\cdot k+1})\dots$.

Using (2) and (4) and equalities $t_n[r] = b_n[n-r], s_n[r] = f_n[n-r]$ (or equating coefficients of x^{r+1} in (6), (7)) we obtain:

$$t_n[r] = q^{2\cdot r-1} t_{n-1}[r-1] - ((q+1)\cdot q^{2\cdot r} - 2\cdot q^r)\cdot t_{n-1}[r] + (q^{2\cdot r+2} - 2\cdot q^{r+1} + 1)\cdot t_{n-1}[r+1], \quad (17)$$

$$f_n[r] = q^r \cdot f_{n-1}[r-1] + (1-q)\cdot q^r \cdot f_{n-1}[r] + (1-q^{r+1})\cdot f_{n-1}[r+1]. \quad (18)$$

Consider the ratios $\tilde{f}_i[r] = \frac{f_i[r]}{f[r]}$ and $\tilde{t}_i[r] = \frac{t_i[r]}{t[r]}$.

Lemma 3 *For each value of i $\tilde{f}_i[r], \tilde{t}_i[r]$ do not increase.*

Proof 8 *We will use mathematical induction on i. Base case: $i = 0$ we obtain: $\tilde{f}_i[r] = \tilde{f}_i[1] = 0, r > 0$. Consider the induction step.*

Let $\tilde{f_{i-1}}[r] = A, \tilde{f_{i-1}}[r+1] = B, B \leqslant A$. We have:

$$\tilde{f}_i[r] - \tilde{f}_i[r+1] = \frac{\tilde{f}_i[r-1]\cdot q^r \cdot f[r-1] + \tilde{f}_i[r]\cdot(1-q)\cdot q^r \cdot f[r] + \tilde{f}_i[r+1]\cdot(1-q^{r+1})\cdot f[r+1]}{f[r]} -$$

$$- \frac{\tilde{f}_i[r]\cdot q^{r+1}\cdot f[r] + \tilde{f}_i[r+1]\cdot(1-q)\cdot q^{r+1}\cdot f[r+1] + \tilde{f}_i[r+2]\cdot(1-q^{r+2})\cdot f[r+2]}{f[r+1]} \geqslant$$

$$\geqslant \frac{A\cdot q^r \cdot f[r-1] + A\cdot(1-q)\cdot q^r \cdot f[r] + B\cdot(1-q^{r+1})\cdot f[r+1]}{f[r]} -$$

$$- \frac{A\cdot q^{r+1}\cdot f[r] + B\cdot(1-q)\cdot q^{r+1}\cdot f[r+1] + B\cdot(1-q^{r+2})\cdot f[r+2]}{f[r+1]} = I.$$

From (7) multiplicating by $x-1$ and considering coefficients of x^{r+1} we obtain equality $f[r] = q^r \cdot f[r-1] + (1-q)\cdot q^r \cdot f[r] + (1-q^{r+1})\cdot f[r+1]$ and using equality (11) $\frac{f[r+1]}{f[r]} = p^{r+1} - 1$ we have:

$$I = (A-B)(q^r\cdot(p^r-1) + (1-q)\cdot q^r - \frac{q^{r+1}}{p^{r+1}-1}) \geqslant 0.$$

Step is proved. Proof for $\tilde{t}_i[r]$ is similar. □

Statement 2 *$t_n[r], f_n[r]$ converge to the $t[r], f[r]$, when n tends to the infinity.*

Proof 9 *From Lemma 3, (17) and (18) we have:*

$$\tilde{f}_i[0] = (1-q)\cdot \tilde{f_{n-1}}[0] + (1-q)\cdot \frac{q}{(1-q)}\tilde{f_{n-1}}[1] \leqslant \tilde{f_{n-1}}[0],$$

$$\tilde{t_i}[0] = -((q+1)-2)\cdot \tilde{t_{n-1}}[0] + (q-1)^2\cdot \frac{q}{(1-q)^2}\tilde{t_{n-1}}[1] \leqslant \tilde{t_{n-1}}[0],$$

therefore $\tilde{t_n}[0]$ *and* $\tilde{f_n}[0]$ *decrease according to* i, *and hence converge to some limits* $\tilde{f}$ *and* $\tilde{t}$, *hence* $t_n[0]$ *and* $f_n[0]$ *have limits. Now, using mathematical induction on* r, *we will prove that* $t_n[r]$ *and* $f_n[r]$ *converge to some limits. Base case for* $r=0$ *is proved. Let prove the induction step. From (17),(18) we have:*

$$(1-q^{r+1})\cdot f_{n-1}[r+1] = f_n[r] - q^r\cdot f_{n-1}[r-1] - (1-q)\cdot q^r\cdot f_{n-1}[r]$$

$$(q^{2\cdot r+2} - 2\cdot q^{r+1} + 1)\cdot t_{n-1}[r+1] = t_n[r] - q^{2\cdot r-1}t_{n-1}[r-1] + ((q+1)\cdot q^{2\cdot r} - 2\cdot q^r)\cdot t_{n-1}[r]$$

Whence , convergence of the right hand side, we obtain that left side also converges. Step is proved. Limits satisfy (8) and (9), hence are equal to $c_1\cdot t[r], c_2\cdot f[r]$ *Because of the choice of* t_q *and* c_q *constants are equal to* 1. $\square$

Following results were computationly obtained, if $p=2$: $t[0]\approx 0.2887881, t[1]\approx 0.5775762$, $t[2]\approx 0.1283503, t[3]\approx 0.0052388$. Also we have $t[i]\leqslant c\cdot q^{i^2}$, for some constant c.

Lemma 4

$$t_n[r] = q^{r^2}\cdot \frac{(\prod_{i=r+1}^{n}(1-q^i))^2}{\prod_{i=1}^{n-r}(1-q^i)} \tag{19}$$

Proof 10 *We will prove using mathematical induction on* n. *Base case for* $n=0$ $t_0[0]=1$. *Let prove the induction step, from (17):*

$t_n[r] = q^{2\cdot r-1}\cdot t_{n-1}[r-1] - ((q+1)\cdot q^{2\cdot r} - 2\cdot q^r)\cdot t_{n-1}[r] + (q^{2\cdot r+2} - 2\cdot q^{r+1} + 1)\cdot t_{n-1}[r+1] =$

using induction hypothesis:

$$= q^{2\cdot r-1}\cdot q^{(r-1)^2}\cdot \frac{(\prod_{i=r}^{n-1}(1-q^i))^2}{\prod_{i=1}^{n-r}(1-q^i)} - ((q+1)\cdot q^{2\cdot r} - 2\cdot q^r)\cdot q^{r^2}\cdot \frac{(\prod_{i=r+1}^{n-1}(1-q^i))^2}{\prod_{i=1}^{n-r-1}(1-q^i)} +$$

$$+(q^{2\cdot r+2} - 2\cdot q^{r+1} + 1)\cdot q^{(r+1)^2}\cdot \frac{(\prod_{i=r+2}^{n-1}(1-q^i))^2}{\prod_{i=1}^{n-r-2}(1-q^i)} = q^{r^2}\cdot \frac{(\prod_{i=r+1}^{n-1}(1-q^i))^2}{\prod_{i=1}^{n-r}(1-q^i)}\cdot$$

$$\cdot((1-q^r)^2 - (1-q^{n-r})\cdot((q+1)\cdot q^{2r} - 2\cdot q^r) + q^{2\cdot r+1}\cdot(1-q^{n-r})\cdot(1-q^{n-r-1})) =$$

$$= q^{r^2}\cdot \frac{(\prod_{i=r+1}^{n}(1-q^i))^2}{\prod_{i=1}^{n-r}(1-q^i)}.$$

Lemma is proved. $\square$.

Statement 3 *When* $p=2$ *at a fixed* r_0 *not equal to* 0 *sequence* $t_n[r_0]$ *increases, when* $n = r_0, r_0+1, \ldots$ *and decreases in case* $r_0 = 0$.

Proof 11 *From Lemma 4:*

$$\frac{t_{n+1}[r_0]}{t_n[r_0]} = \frac{(1-q^{n+1})^2}{(1-q^{n+1-r_0})} = \frac{1-q^n+q^{2\cdot n+2}}{1-q^{n+1-r_0}}.$$

When $r_0 = 0$ *we have* $q^{n+1} + q^{2\cdot n+2} < q^n$, *hence ratio is smaller than 1 and sequence* $t_n[0]$ *decreases. When* $r_0 \geqslant 1$ *we have* $q^{n+1-r_0} + q^{2\cdot n+2} > q^n$, *hence ration is bigger than 1 and sequence* $t_n[r_0]$ *increases.* $\square$

3 Distribution of stohastic variable ζ

Consider following stohastic process. On the zero step matrix A_0 over Z_2 with size $n \times n$ is constructed, each element of A_0 is 0 or 1 with probability $\frac{1}{2}$. With the help of elementary transformations of the columns of matrix A_0 we reduce it to the form where left columns are linearly independent and the rest are zeros. On the first step we construct A_1 randomly filling zero-columns of A_0, and reducing it to the form with $corankA_1$ zero-columns at the end. Similarly we construct $A_2, A_3 \dots A_i \dots$ filling zero-columns of the previous matrix and reducing it to the form with $corank$ zero-columns at the end.

Denote for $\zeta = min\{i|corankA_i = 0\}$. Constructed stohastic variable is identical to ζ from [2]. Note, that if corank of A_i is equal to k, then considering columns of A_{i+1} in basis, which contains first $n-k$ colums of A_i, we obtain that corank of A_{i+1} is distributed as corank of random matrix with size $k \times k$. Denote for $\mu_{n,k} = P(\zeta = k)$. Then $\mu_{n,0} = t_n[0]$,

$$\mu_{n,k} = \Sigma_{i=1}^{n} t_n[i] \cdot \mu_{i,k-1}. \tag{20}$$

Now we obtain lower and upper bounds for $\mu_{n,k}$.

$$\mu_{n,k} = t_n[n]^k \cdot t_n[0] + \Sigma_{i=0}^{k-1}\Sigma_{j=1}^{n-1} t_n[n]^i \cdot t_n[j] \cdot \mu_{j,k-i-1}.$$

Where i is number of zero-matrices, and j is a corank of the first none-zero. Then $\mu_{1,k} = (\frac{1}{2})^{k+1}$.

$$\mu_{2,k} = \frac{3}{8} \cdot (\frac{1}{16})^k + \frac{9}{16} \cdot \Sigma_{i=0}^{k-1} (\frac{1}{16})^i \cdot (\frac{1}{2})^{k-i} =$$

$$= \frac{3}{8} \cdot (\frac{1}{16})^k + \frac{9 \cdot ((\frac{1}{2})^k - (\frac{1}{16})^k)}{32 \cdot (\frac{1}{2} - \frac{1}{16})} = \frac{9}{14} \cdot (\frac{1}{2})^k - \frac{15}{56} \cdot (\frac{1}{16})^k.$$

Similarly we have

$$\mu_{3,k} = \frac{7}{10} \cdot (\frac{1}{2})^k - \frac{105}{248} \cdot (\frac{1}{16})^k + \frac{511}{9920} \cdot (\frac{1}{512})^k.$$

Now we obtain lower and upper bounds for $\mu_{n,k}, n > 3$. We do this by selecting the first moment when corank is lesser or equal 3. From Statement 3 $\alpha = \Sigma_{i=4}^{\infty} t[i]$ - is upper bound for probability of random matrix to have corank greater than 3. Using Statement 3 we obtain:

$$\mu_{n,k} \leqslant (\alpha)^k \cdot t_4[0] + \Sigma_{i=0}^{k-1}\Sigma_{j=1}^{3} (\alpha)^i \cdot t[j] \cdot \mu_{j,k-i-1} \leqslant$$

Replacing $\mu_{1,k}, \mu_{2,k}, \mu_{3,k}$ by obtained values and using known tresholds for t we have :

$$\leqslant 0.75001 \cdot (\frac{1}{2})^k - 0.58599 \cdot (\frac{1}{16})^k + 0.14156 \cdot (\frac{1}{512})^k + 0.00207 \cdot (\alpha)^k, \tag{21}$$

$$\alpha = 1 - \Sigma_{i=0}^{4} t[i] \leqslant 0.0000467,$$

$$\mu_{n,k} \geqslant (t_4[4])^k \cdot t[0] + \Sigma_{i=0}^{k-1}\Sigma_{j=1}^{3} (t_4[4])^i \cdot t_4[j] \cdot \mu_{j,k-i-1} \geqslant$$

Replacing $\mu_{1,k}, \mu_{2,k}, \mu_{3,k}$ by obtained values and using known thresholds for t we have :

$$\geqslant 0.72580 \cdot (\frac{1}{2})^k - 0.50404 \cdot (\frac{1}{16})^k + 0.09126 \cdot (\frac{1}{512})^k - 0.02425 \cdot (\frac{1}{65536})^k \tag{22}$$

hence when $k \geqslant 2$ we obtain:

$$(0.72580 - 0.50404 \cdot \frac{1}{64}) \cdot (\frac{1}{2})^k = 0.717924 \cdot (\frac{1}{2})^k \leqslant \mu_{n,k} \leqslant 0.75001 \cdot (\frac{1}{2})^k. \tag{23}$$

Now we will found bounds for $\Sigma_k = -\Sigma_{i=k}^{\infty} ln(P(\zeta \leqslant i))$ (13), auxillary value, which we will use later. From (23):

$$1 - 0.75001 \cdot (\frac{1}{2})^k \leqslant 1 - P(\zeta > k) =$$

$$= P(\zeta <= k) \leqslant 1 - (0.72580 - 0.50404 \cdot \frac{1}{64}) \cdot (\frac{1}{2})^k \leqslant 1 - 0.717924 \cdot (\frac{1}{2})^k, k \geqslant 2, \tag{24}$$

$$0.62036 \leqslant P(\zeta \leqslant 1) \leqslant 0.62744. \tag{25}$$

Then when $k \geqslant 2$ we have:

$$\Sigma_k = -\Sigma_{i=k}^{\infty} ln(P(\zeta \leqslant i)) = \Sigma_{i=k}^{\infty} \Sigma_{j=1}^{\infty} \frac{(1 - P(\zeta \leqslant i))^j}{j} = \Sigma_{j=1}^{\infty} \frac{\Sigma_{i=k}^{\infty} (1 - P(\zeta \leqslant i))^j}{j}.$$

We obtain the bounds:

$$\Sigma_k \geqslant \Sigma_{j=1}^{\infty} \frac{\Sigma_{i=k}^{\infty} (0.717924)^j \cdot (\frac{1}{2})^{i \cdot j}}{j} = \Sigma_{j=1}^{\infty} \frac{(0.717924 \cdot \frac{1}{2^k})^j}{j \cdot (1 - \frac{1}{2^j})} \geqslant$$

$$\geqslant 2 \cdot (0.717924) \cdot \frac{1}{2^k} + \frac{4}{3} \cdot (0.717924)^2 \cdot \frac{1}{4^k}, \tag{26}$$

$$\Sigma_k \leqslant \Sigma_{j=1}^{\infty} \frac{\Sigma_{i=k}^{\infty} (0.75001)^j \cdot (\frac{1}{2})^{i \cdot j}}{j} = \Sigma_{j=1}^{\infty} \frac{(0.75001 \cdot \frac{1}{2^k})^j}{j \cdot (1 - \frac{1}{2^j})} \leqslant$$

$$\leqslant 2 \cdot (0.75001) \cdot \frac{1}{2^k} + \frac{4}{3} \cdot (0.75001)^2 \cdot \frac{1}{4^k} \cdot \frac{1}{1 - (0.75001) \cdot \frac{1}{2^k}}. \tag{27}$$

4 Expected value and distribution of auxillary stohastic variable τ

There was shown in [1] , that for $r > t - 1 > 0$ fulfillment of inequalities

$$\zeta_r \leqslant t - 1, \zeta_{r-1} \leqslant t - 1, \zeta_{r-2} \leqslant t - 2, \ldots, \zeta_{r-t+1} \leqslant 1$$

is sufficiently, for constructing approximation with number $r + 1$ with the help of coefficients of approximations with numbers $r, r - 1, \ldots, r - t$. For simplification we denote

$$\xi_1 = \zeta_r, \ldots, \xi_t = \zeta_{r-t+1}, \ldots.$$

Let:

$$\theta = min\{t \geqslant 1 : \xi_1 \leqslant t - 1, \xi_i \leqslant t - i + 1, i = 2, \ldots, t\},$$

$$\tau = min\{t \geqslant 1 : \xi_i \leqslant t - i + 1, i = 1, \ldots, t\}.$$

Let $\Pi(i) = \Pi_{k=1}^{i} P(\xi \leqslant k), \Pi(0) = 1$, and $f(s)$ is generating function of distribution of τ.

Lemma 5

$$P(\tau = k) = \Sigma_{n,a_i \in N \cup \{0\}, a_1 + \ldots + a_n = k} (-1)^{n-1} \cdot \Pi_{i=1}^{n} \Pi(a_i), k \in N \tag{28}$$

Proof 12 *We will prove using mathematical induction on* k. *Base case for* $k = 1$: $P(\tau = 1) = P(\xi \leqslant 1) = \Pi(1)$. *Induction step:*

$$P(\tau = k) = P(\{\xi_i \leqslant k-i+1, i = 1, 2, \ldots, k\} \backslash \bigcup_{j=1}^{k-1} \{\tau = j, \xi_{j+i} \leqslant k-j-i+1, i = 1, 2, \ldots, k-j\}) =$$

(it is true, that $\xi_i \leqslant k - i + 1, i = 1, 2, \ldots, k$, *but it is not true that* $\tau < k$.*)*

$$= \Pi(k) - \Sigma_{j=1}^{k-1} \Pi(k-j) \cdot P(\tau = j) = \Pi(k) +$$

$$+ \Sigma_{a_{n+1}=1}^{k-1} (-\Pi(a_{n+1})) \cdot \Sigma_{n,a_i \in N \cup \{0\}, a_1 + \ldots + a_n = k - a_{n+1}} (-1)^{n-1} \cdot \Pi_{i=1}^{n} \Pi_{a_i} =$$

$$= \Sigma_{n,a_i \in N \cup \{0\}, a_1 + \ldots + a_n = k} (-1)^{n-1} \cdot \Pi_{i=1}^{n} \Pi(a_i).$$

Lemma is proved. □

Statement 4

$$\frac{1}{1 - f(s)} = \Sigma_{i=0}^{\infty} \Pi(i) \cdot s^i, 0 < s < 1 \tag{29}$$

Proof 13 *Using the fact, that* $1 - \frac{1}{1+x} = \Sigma_{i=1}^{\infty} (-1)^{i-1} \cdot s^i$ *we obtain:*

$$1 - \frac{1}{1 + \Sigma_{i=1}^{\infty} \Pi(i) \cdot s^i} = \Sigma_{j=1}^{\infty} (-1)^{j-1} (\Sigma_{i=1}^{\infty} \Pi(i) \cdot s^i)^j =$$

$$= \Sigma_{j=1}^{\infty} \Sigma_{n,a_i \in N \cup \{0\}, a_1 + \ldots + a_n = k} (-1)^{n-1} \cdot \Pi_{i=1}^{n} \Pi(a_i) \cdot s^j = f(s)$$

when $0 < s < 1$. *Whence we obtain the required equality.* □

Theorem 2

$$2.382189 \leqslant M\tau \leqslant 2.484705 \tag{30}$$

$$12.797192 \leqslant M\tau^2 \leqslant 14.300803 \tag{31}$$

Proof 14 *Let* $\Pi(\infty) = \Pi_{i=1}^{\infty} P(\xi \leqslant i)$. *Because*

$$\Sigma_i = -ln(\Pi(\infty)) + ln(\Pi(i-1)), \tag{32}$$

then using (25), (27) (26) we obtain $0.402462 \leqslant \Pi(\infty) \leqslant 0.419781$. *We have, that if* $s \to 1$ *then right hand side tends to infinity, hence* $f(1) = 1$. *Then, multiplicating both parts of (29) by* $1 - s$ *we have:*

$$\frac{1-s}{f(1) - f(s)} = 1 + \Sigma_{i=1}^{\infty} (\Pi(i) - \Pi(i-1)) \cdot s^i. \tag{33}$$

When $s \to 1$ *we have:*

$$f'(1)^{-1} = \Pi_{\infty}. \tag{34}$$

Hence we obtain lower and upper bounds for $M\tau = f'(1)$: $2.382189 \leqslant M\tau \leqslant 2.484705$.

From

$$(1-s)\cdot\Sigma_{i=0}^{\infty}(\Pi(i)-\Pi(\infty))\cdot s^i = 1-\Pi(\infty)+\Sigma_{i=1}^{\infty}(\Pi(i)-\Pi(i-1)),$$

we have:

$$\frac{\frac{1-s}{f(1)-f(s)}-f'(1)^{-1}}{1-s} = \Sigma_{i=0}^{\infty}(\Pi(i)-\Pi(\infty))\cdot s^i$$

reducing left side to the common denominator and proceeding to the limit, when $s\to 1$

$$\frac{f''(1)}{2\cdot f'(1)^2} = \Sigma_0^{\infty}(\Pi(i)-\Pi(\infty)).$$

Now from (32) and (34) we have:

$$f''(1) = 2\cdot f'(1)\cdot(\Sigma_{i=1}^{\infty}exp(\Sigma_i)-1). \tag{35}$$

From (34) and (35) and using Taylor series for exponent:

$$f''(1) = 2\cdot f'(1)\cdot((f'(1)-1)+\Sigma_{i=2}^{\infty}\Sigma_{j=1}^{\infty}\frac{(\Sigma_i)^j}{j!}).$$

From (26) and (27) we know, that $c_1\cdot 2^{-i+1}\leqslant\Sigma_i\leqslant c_2\cdot 2^{-i+1}$, *where* $c_1=0.71792, c_2=0.77884$. *Now from (30) we obtain inequalities*

$$f''(1)\geqslant 6.58527+4.764378\cdot\Sigma_{k=1}^{\infty}(0.71792\cdot 2^{-k+1}+0.5\cdot(0.71792)^2\cdot 4^{-k+1})\geqslant 10.415003,$$

$$f''(1)\leqslant 7.378108+4.96941\cdot\Sigma_{k=1}^{\infty}(0.77884\cdot 2^{-k+1}+0.5\cdot(0.77884)^2\cdot 4^{-k+1}+\frac{(0.77884)^3\cdot 8^{-k+1})}{6\cdot(1-(0.77884)^3\cdot 0.25)}\leqslant$$

$$\leqslant 11.816098.$$

$M\tau^2 = f''(1)+f'(1)$, *using (30) we obtain required inequality.* □

Now we will obtain bounds for values of all derivatives. Let $\gamma(s)=\frac{f(s)-1}{s-1}, g(s)=1-\Sigma_{i=1}^{\infty}\Pi(i-1)\cdot P(\zeta>i)$. From (33)

$$\frac{1-s}{1-f(s)} = 1-\Sigma_{i=1}^{\infty}\Pi_{i-1}\cdot P(\zeta>i)\cdot s^i,$$

$$\gamma(s)\cdot g(s) = 1.$$

Differentiating this equality n times:

$$\Sigma_{i=0}^{n}\binom{n}{i}\cdot\gamma^{(n-i)}(s)\cdot g^{(i)}(s) = 0,$$

$$\gamma^{(n)}(s) = \frac{-\Sigma_{i=1}^{n}\binom{n}{i}\cdot\gamma^{(n-i)}(s)\cdot g^{(i)}(s)}{g(s)}. \tag{36}$$

Note, that $g^{(i)}(s)<0<g(s)$, when $0\leqslant s\leqslant 1, i\geqslant 1$. Hence, if for some functions $g_1(s), g_2(s)$ the following statements are fullfilled:

$$g_1(s_0) = g_2(s_0) = g(s_0), \tag{37}$$

$$g_1^{(i)}(s_0) \leqslant g^{(i)}(s_0) \leqslant g_2^{(i)}(s_0) < 0, i \geqslant 1, \tag{38}$$

then

$$(\frac{1}{g_2})^{(i)}(s_0) \leqslant \gamma^{(i)}(s_0) \leqslant (\frac{1}{g_1})^{(i)}(s_0), i \geqslant 1. \tag{39}$$

We take g_i equal to $\alpha_i - \frac{\beta_i}{1-\frac{s}{2}}, \alpha_i = g(s_0) + \frac{\beta_i}{1-\frac{s_0}{2}}$,for satisfiyng (37) and $\beta_1 = 0.75001$, and $\beta_2 == 0.28893 \leqslant 0.71792 \cdot \Pi(\infty)$,is sufficient due to the defenition of $g(s)$ and (24) to satisfy (38) . Then

$$(\frac{1}{g_i})^{(n)}(s_0) = (\frac{1-\frac{s}{2}}{\alpha_i - \beta_i - \frac{\alpha_i \cdot s}{2}})^{(n)}|_{s=s0} = n! \cdot \frac{2 \cdot \beta_i \cdot \alpha_i^n}{\alpha_i \cdot (2 \cdot (\alpha_i - \beta_i) - \alpha_i \cdot s_0)^{n+1}}, n \geqslant 1.$$

We obtain, taking $\alpha_i, \beta_i, s_0 \in \{0, 1\}$

$$0.22416 \cdot (0.64446)^n \cdot n! \leqslant \gamma^{(n)}(0) \leqslant 0.42857 \cdot (0.87501)^n \cdot n! \tag{40}$$

$$1.37657 \cdot (2.37657)^n \cdot n! \leqslant \gamma^{(n)}(1) \leqslant 1.95908 \cdot (4.72711)^n \cdot n! \tag{41}$$

Whence we obtain bounds for derivatives of function $f(s)$.

$$1.37657 \cdot (2.37657)^{n-1} \cdot n! \leqslant f^{(n)}(1) \leqslant 1.95908 \cdot (4.72711)^{n-1} \cdot n!$$

Now we will find bounds for $P(\tau > n)$.

$$f^{(n)}(0) = i \cdot \gamma^{(n-1)}(0) + \gamma^{(n)},$$

$$P(\tau = n) = \frac{f^{(n)}(0)}{n!} = \frac{\gamma^{(n-1)}}{(n-1)!} - \frac{\gamma^{(n)}}{n!},$$

hence:

$$P(\tau > i) = \frac{\gamma^{(i)}}{i!}. \tag{42}$$

Now from (40) we obtain:

$$0.22416 \cdot (0.64446)^i \leqslant P(\tau > i) \leqslant 0.42857 \cdot (0.87501)^i. \tag{43}$$

Here we can obtain more precise upper bounds for $P(\tau > i)$, using polynomials of higher degree. For example consider function $\tilde{g_1}(s) = \alpha - \beta \cdot s - \frac{\delta}{1-\frac{s}{2}}$, where $\alpha = 1+\delta,\ \delta = 0.47060, \beta = 0.14434 = 0.37964 - 0.23530 > g'(0) - 0.5 \cdot \delta$. Then $\tilde{g_1}(0) = g(0)$ and $\tilde{g_1}^{(n)} \leqslant g^{(n)}$, and hence $\gamma^{(n)}(0) < \frac{1}{\tilde{g_1}}^{(n)}(0)$. From the other side:

$$\frac{1}{\tilde{g_1}(s)} = \frac{2-s}{\beta \cdot s^2 - (2 \cdot \beta + \alpha) \cdot s + 2} = \frac{c_1}{s_1 - s} + \frac{c_2}{s_2 - s}$$

wher s_1, s_2 are roots of the polynomial in denominator, $c_1 = \frac{2-s_1}{\beta \cdot (s_2 - s_1)}, c_2 = \frac{2-s_2}{\beta \cdot (s_1 - s_2)}$. Finding s_1 and s_2 we obtain:

$$\gamma^{(n)}(0) < n! \cdot (c_1 \cdot s_1^{-n} + c_2 \cdot s_2^{-n}) \leqslant n!(0.52487 \cdot (0.78807)^{n+1} + 6.40328 \cdot (0.09157)^{n+1}).$$

Similarly taking $\tilde{\tilde{g}}_1(s) = \alpha - \beta \cdot s - \varepsilon \cdot s^2 - \frac{\delta}{1-\frac{s}{2}}$ we have:

$$\frac{1}{\tilde{\tilde{g}}_1(s)} = \frac{2-s}{2 - 1.7593 \cdot s + 0.14435 \cdot s^2 + 0.21115 \cdot s^3}$$

and upper bound:

$$P(\tau > n) = \frac{\gamma^{(n)}(0)}{n!} < 0.41973 \cdot (0.76773)^n + 0.19412 \cdot (0.07398)^n + 0.38590 \cdot (0.18590)^n. \quad (44)$$

5 Expected value of stochastic variable θ

Now we will found lower and upper bounds for expected value of θ.

Theorem 3

$$3.63148 \leqslant M\theta \leqslant 3.84696 \quad (45)$$

$$22.33992 \leqslant M\theta^2 \leqslant 29.58587 \quad (46)$$

Proof 15 *Let ω be an event, in which $\zeta_1 = i$. Then, if $\zeta_k \leqslant i - k + 1, k = 2, \ldots, i$, then $\theta = i + \tilde{\tau} = \tau(\omega) + \tilde{\tau}$, where $\tilde{\tau}, \tau$ are variables identically and independently distributed, in other case $\theta = \tau$. We have:*

$$M\theta = M\tau \cdot (1 + \Sigma_{i=1}^{\infty} P(\zeta = i) \cdot \Pi(i-1)) \quad (47)$$

$$\frac{M\theta}{M\tau} \geqslant 1 + P(\zeta = 1) + P(\zeta = 2) \cdot \Pi(1) + P(\zeta = 3) \cdot \Pi(2) + P(\zeta > 3) \cdot \Pi(\infty)$$

$$\frac{M\theta}{M\tau} \leqslant 1 + P(\zeta = 1) + P(\zeta = 2) \cdot \Pi(1) + P(\zeta = 3) \cdot \Pi(2) + P(\zeta > 3) \cdot \Pi(3)$$

$$3.63148 < 1.52443 M\tau < M\theta < 1.54826 M\tau < 3.84696$$

Now we find bounds for $M\theta^2$. Let I_1 be a stohastic variable equal to 1, if $\exists k | \zeta_1 = k, \zeta_i \leqslant k - i + 1, i = 2, \ldots, k$ and 0 in other case. Then $\theta = \tau + I_1 \cdot \tilde{\tau}$. Note, that I_1 and $\tilde{\tau}$ are independent variables. We have:

$$M\theta^2 = M\tau^2 + 2 \cdot M\tau \cdot \tilde{\tau} \cdot I_1 + M(\tilde{\tau} \cdot I_1)^2 =$$

$$= M\tau^2(1 + MI_1^2) + 2 \cdot M\tau \cdot I_1 \cdot M\tau = M\tau^2(1 + MI_1) + 2 \cdot M\tau \cdot M\tau \cdot I_1 \quad (48)$$

Note, that $1 + MI_1 = \frac{M\theta}{M\tau}$. Hence, it's enough to find bounds for $M\tau \cdot I_1$.

$$M\tau \cdot I_1 = \Sigma_{i=1}^{\infty} i \cdot P(\zeta = i) \cdot \Pi(i-1).$$

Using (21) and (22) we obtain:

$$M\tau \cdot I_1 \leqslant \Sigma_{i=1}^{\infty} i \cdot 0.75001 \cdot (\frac{1}{2})^i = 1.50002 \quad (49)$$

$$M\tau \cdot I_1 \geqslant \Sigma_{i=1}^{\infty} i \cdot (0.72580 \cdot (\frac{1}{2})^i - 0.50404 \cdot (\frac{1}{16})^i) \cdot \Pi(\infty) = 1.41575 \cdot (M\tau)^{-1} \quad (50)$$

At the end using (30), (31), (49),(50), inequalities for MI_1 and equality (48) we have:

$$22.33992 \leqslant M\theta^2 \leqslant 29.58587$$

□.

6 Upper bound for algorithm's memory requirements

For the work of the algorithm from [1], we have to keep θ previous approximations to construct the new one. Further we talk about probabilities under conditition of successfully completion of first r steps, where r — number of stored approximations.

Theorem 4 *For successful execution of the algorithm with probability 0.99 on matrices of size 2^s and block-size 2^k, where $(s-k) > 10$, is sufficient to keep $2.622407 \cdot (s-k) + 18.805443$ previous approximations.*

Proof 16 *We will find the number of approximations n for successful execution of l steps of algorithm with probability bigger than 0.99. This probability will be bigger or equal to $1 - l \cdot P(\theta > n)$. Hence it is sufficient, that $P(\theta > n) < \frac{0.01}{l}$. From (21) and (44):*

$$P(\theta > n) = P(\tau > n) + \Sigma_{i=1}^{n-1} P(\zeta = i) \cdot \Pi(i-1) \cdot P(\tau > n-i), \tag{51}$$

$$P(\theta > n) \leqslant P(\tau > n) + 0.33866 \cdot P(\tau > n-1) +$$

$$+0.47059 \Sigma_{i=2}^{n-1} 2^{-i} \cdot (0.41973 \cdot (0.76773)^{n-i} + 0.19412 \cdot (0.07398)^{n-i} + 0.38590 \cdot (0.18590)^{n-i}) \leqslant$$

$$\leqslant 0.90382 \cdot (0.76773)^{n-1} + 0.06601 \cdot (0.07398)^{n-1} + 0.33866 \cdot (0.5)^{n-1} + 0.23154 \cdot (0.18590)^{n-1}.$$

Number of steps of the algorithm is equal to $l = 2^{s-k}$. n will be large, so $P(\theta > n) < 0.90383 \cdot (0.76773)^{n-1}$, and it is true, if $n = 2.622407 \cdot (s-k) + 18.805443$. □

Theorem 5 *For successful execution of the algorithm with probability 0.99 on matrices of size 2^s and block-size 2^k, where $(s-k) > 10$, is necessary to keep $1.3 \cdot (s-k) + 6.22$ previous approximations.*

Proof 17 *Let r be the number of stored approximations. Then $\{\theta_i > r\}$ and $\{\theta_{i+r} > r\}$ are independent events. Than probability of successful execution of the algorithm is not bigger than $1 - P(\theta > r) \cdot \frac{2^{s-k}}{r}$. Using (51), (22) and (43) we obtain:*

$$P(\theta > n) \geqslant P(\tau > n) + 0.33139 \cdot P(\tau > n-1) + 0.05947 \cdot P(\tau > n-2) \geqslant$$

$$\geqslant 0.22416 \cdot (0.64446)^n \cdot (1 + 0.51421 + 0.14318) = 0.37152 \cdot (0.64446)^n.$$

Hence, when $r < 1.3 \cdot (s-k) + 6.22$, $1 - P(\theta > r) \cdot \frac{2^{s-k}}{r} < 0.99$. □

7 Computed results

Obtained probabilities can be computed for $n = 64$ using computer. Distribution of coranks of matrices with size non-greater than 64 we compute using formulas from the first part. Functions $\mu_{64,k}$ we compute from (20), up to $k \leqslant 64$. Further members of the product from the right side of formula $M\tau = (\Pi(\infty))^{-1}$ can be discarded, because their product differs from 1 lesser than $\approx 2^{-63}$. We have $M\tau \approx 2.39$. To compute $M\theta$ we use (47): $M\theta \approx 3.67$. Then using (29) and (51), we compute $P(\tau > k), P(\theta > k)$ and obtain that, for $k > 30$ $P(\tau > k) \approx 0.13816 \cdot (0.75471)^{k+1}$, $P(\theta > k) \approx 0.26795 \cdot (0.75471)^{k+1}$, hence for $k = 60$ with probability bigger than 0.99 algorithm will be successfully executed. 60 will require less than

30 gigabyte of memory.

In the proof of theorem 4, expression $1 - l \cdot P(\theta > k)$ was used to find lower bound for probability of successful execution. But $P(\theta > k)$ will not be summarized, and this probability will be higher. Also on the first step, because of the small size of approximation polynomials we can keep much more approximations. Compute the probability, that using m GB of memory, algorithm for matrix of size N and block-size $n > 64$ will be successfully executed. We will fix K - maximal number of kept approximations. Let $g[i][j]$ be the probability of successfully execution of i steps with the condition $\theta_i = j$. $g[0][0] = 1.0$. Let find $g[i][k]$. Memory restriction on the number of kept approximations l_i on the i-th step is $i-1+i-2+\ldots+i-l_i \leqslant \frac{m \cdot 2^3 3}{s^2}$. $l_i(i - \frac{(l_i+1)}{2}) \leqslant \frac{m}{s^2}$, whence find l_i. Let $l_i = min(l_i, K)$. Then run through $k = 1, \ldots, l_i$. If $\theta_i = k$ one of the inequalities from the definition of θ has to turn in equality. If it is not the first inequality, than let it be with number $k - j + 1$. Then probability of this event is equal to

$$P(\zeta = j) \cdot \Pi(j-1) \cdot (P(\zeta \leqslant k-2) \cdot \Pi_{i=j+1}^{k-1} P(\zeta \leqslant i-1) - P(\theta \leqslant k-j | \zeta_i \leqslant k-i, i = 2, \ldots k-j))$$

. Note, that for $i - j + 1 \leqslant h \leqslant i$, will be sufficient memory, and if it will be sufficient for $i - j$ - th step, it will be sufficient for all steps with numbers from $i - k + 1$ to $i - j - 1$. Hence $\theta_{i-k} \leqslant l_{i-j} - j$, summarizing obtained probabilities $\Sigma_{z=0}^{l_{i-j}-j} g[i-j][z]$ we obtain the answer in this case. In the case when $\zeta_i = k - 1$, we will also focus on the case of first equality:

$$P(\zeta = k-1) \cdot (\Sigma_{z=1}^{k-1} \Pi(z-1) \cdot P(\zeta = z) \cdot \Pi_{j=z+1}^{k-1} P(\zeta \leqslant j-1) \cdot \Sigma_{p=0}^{l_{i-z}-z} g[i-k][p] +$$
$$+ \Pi(k-1) \cdot P(\zeta = 0) \cdot \Sigma_{p=0}^{n} g[i-k][p]).$$

Now we compute for $\frac{N}{n}$ steps, under condition of successful execution of first $g \approx 100$ steps we obtain the probability of successful execution of the algorithm. Using this algorithm was obtained, that for matrix of size $N = 2^{26}$ and block-size $n = 2^6$ is sufficient $m = 26$ GB of memory, and block-size $n = 2^{11}$ $m = 650$ GB of memory.

8 Conclusion

Algorithm from [1] was known, but there wasn't (or was but not very accurate) theoretically estimates of its efficiency. In this work the lower and upper bound for the expected number of previous approximations is obtained:

$$3.63148 \leqslant M\theta \leqslant 3.84696.$$

In the previous works only upper bounds were obtained. In [1] it was some constant C independent from the size of matrix. In [2] the result $M\theta < 7.233$ was obtained. Logarifmically depended on size of matrix lower and upper bounds are found for memory requirements. There were not any similar results before. Method proposed by A.M. Zubkov was used to obtain the results.

Using this results it can be shown that the algorithm from [1] is better (for exmaple by number of operations and memory usage) than other existing algorithms. Distribution of coranks of random (symmetric and none-symmetric) matrices were already known(papers: [3],[4]), but the method proposed in this paper is new and may be used for similar calculations in other cases(for example anti-symmetric matrices).

References

1. *Cherepniov M.A.* Block Lancosh-based algorithm for solving sparse linear systems // Discrete Mathematics (in Russian). 2008. V. 20. N. 1. P. 145-150.

2. *Cherepniov M.A.* Algorithms of constructing Pade-approximations // Materials of the Fourth international scientific conference on the problems of security and counter terrorism. Moscow State university named after M.V. Lomonosov, 30–31 october 2008. V. 2. Materials of the seventh all-russian conference "Mathematics and Security of Information Technologies" (MaSIT-2008). Moscow: MCCME, 2009. P. 21-22.

3. *Goldman J., Rota G.-C.* On the foundations of combinatorial theory IV: Finite vector spaces and Eulerian generating functions // Stud. Appl. Math. 1970. V. 49(3). P. 239-258.

4. *Carlitz L.* Representations by quadratic forms in a finite field // Duke Math. 1954. J. 21. P. 123-137.

Accepted for publication 7.06.2010.

ОЦЕНКА ОЖИДАЕМОГО ВРЕМЕНИ РАБОТЫ И НЕОБХОДИМОГО ОБЪЁМА ПАМЯТИ ДЛЯ УСПЕШНОГО ЗАВЕРШЕНИЯ АЛГОРИТМА РЕШЕНИЯ БОЛЬШИХ РАЗРЯЖЕННЫХ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

© **Василий Вадимович Астахов**

Московский государственный университет им. М.В. Ломоносова, Ленинские горы, 1, Москва, 119991, Россия, кафедра теории чисел, студент механико-математического факультета, e-mail: astvvas88@mail.ru

Ключевые слова: разряженные линейные системы; аппроксимации Паде; распределение корангов.

В работе рассматривается алгоритм решения больших разреженных линейных систем над Z_2, использующий построение матричных аппроксимаций Паде. Предполагается, что элементы аппроксимационных многочленов статистически независимы и равномерно распределены. Строится метод для нахождения распределений корангов для случайных симметричных, кососимметричных и обычных матриц. Даются оценки сверху и снизу на среднее число предыдущих аппроксимаций, необходимых для построения новой аппроксимации. Выявлена логарифмическая зависимость для достаточного числа хранимых аппроксимаций на каждом шагу, для успешного завершения алгоритма с вероятностью 0,99. Средние значения и необходимый объем памяти вычислены при помощи алгоритма, результаты также приведены в работе.

UDC 519.688

PARALLEL COMPUTATION OF LAGRANGE RESOLVENTS BY MULTI-RESOLVENTS

© **Philippe Aubry, Annick Valibouze**
LIP6, UPMC, 4, place Jussieu, F-75252 Paris Cedex 05, France,
e-mail: Philippe.Aubry@upmc.fr, Annick.Valibouze@upmc.fr

Key words: Lagrange resolvent; Galois group; galoisian ideal; triangular ideal; double class; parallel computation.

The goal of this paper is the parallel computation of Lagrange resolvents of a univariate polynomial. The computation of Lagrange resolvents of a univariate polynomial has significance for Galois Theory. Since Lagrange's algorithms, many other algorithms for computing some particular resolvents, called absolute, were developed from the fundamental theorem of symmetric functions. The algebraic algorithms for non absolute resolvents are few and recent because they use galoisian ideals that were introduced recently. However these algorithms become time and space consuming when the degree of the polynomial increases. This motivates their parallelization. Rennert proposed a multi-modular method for computing absolute resolvents of polynomials with integer coefficients. We show that the same techniques can be extended to any resolvent. This method is naturally parallelizable. Moreover, we give a decomposition formula of resolvents which makes possible another level of parallelization. This leads to an algorithm with a doubly parallel character.

1 Introduction

The Lagrange resolvent of a univariate polynomial f is a fundamental tool in Galois theory (see [1] and [2]). It is a univariate polynomial obtained from a multivariate polynomial transformation of f. Its factors are used to describe the action of the Galois group on another group stabilizing the multivariate polynomial Θ used for the transformation; hence, they determine the Galois group of f by using matrices of groups; moreover, the evaluation in Θ of any factor of the resolvent produces a minimal polynomial of a galoisian ideal (see [3]). The parallel method that we describe is inspired by Rennert's work (see [4]) for the restricted case of the resolvents relative to the symmetric group, called absolute resolvents. Nevertheless Rennert's method cannot be adapted automatically to the general case of resolvents relative to subgroups of the symmetric group. The reader can refer to Example 1 that illustrates one of the simplifications existing when the reference group is the symmetric group. This paper does not only extend to any resolvent the multimodular parallelisation proposed by Rennert, but presents another level of parallelisation thanks to a new decomposition formula of resolvents given in Theorem 1. Moreover, the theoretical study that leads to this decomposition, together with the description of a strategy for the parallel computation, bring to the subject greater clarity.

Section 2 introduces galoisian ideals and Lagrange resolvents with some properties. Section 3 establishes Theorem 1 in which the resolvent splits into factors corresponding to double

classes of conjugacy. In Section 4 the polynomial f is supposed to be reducible. We construct the Lagrange resolvent of f from the Lagrange resolvents of its factors following Theorem 1. These latter, called multi-resolvents, may be computed in parallel. Section 5 applies the method of Section 4 to the important case of irreducible (or reducible) polynomials over the rational field. By computing a common denominator of such a polynomial f, we can assume that its coefficients are integers. Since the image of f modulo p is reducible for many prime integers p, we can perform the computation of a Lagrange resolvent of f in $\mathbb{F}_p[x]$ by the above parallel computation with multi-resolvents. The Lagrange resolvent of f in $\mathbb{Z}[x]$ is finally lifted from those in $\mathbb{F}_p[x]$ by using the Chinese Remainder Theorem. This multimodular method is clearly "doubly" parallel since the Lagrange resolvents of f in $\mathbb{F}_p[x]$ are computed independently. Finally, Section 6 is devoted to the parallel algorithm description.

Throughout this paper, k is a perfect field, $\overline{k}$ an algebraic closure of k, f a square-free univariate polynomial of $k[x]$ with degree n and $\underline{\alpha} = (\alpha_1, \ldots, \alpha_n)$ in $\overline{k}^n$ is a tuple of the n distinct roots of f.

General notation For a variety $V \subset \overline{k}^n$, the ideal $Id(V)$ of V is the set of polynomials with coefficients in k vanishing on each element of V. Let I be an ideal of $k[x_1, \ldots, x_n]$, the algebraic variety $V(I)$ of I is the set of point in k^n where every polynomial in I vanishes. The symmetric group of degree n is denoted by $\mathfrak{S}_n$. Given two ideals I and J, the *injector* $Inj(I, J)$ of I in J is the set of elements of $\mathfrak{S}_n$ sending each element of I in J. The subset $Stab_{\mathfrak{S}_n}(I) := Inj(I, I)$ of $\mathfrak{S}_n$ is a group, called the *stabilizer* of I in $\mathfrak{S}_n$ (in literature, it is also called the *decomposition group* of I). For $H < \mathfrak{S}_n$ and $\sigma \in \mathfrak{S}_n$, $H^\sigma = \sigma H \sigma^{-1}$.

2 The Lagrange resolvent

The results not referenced or proved can be found in [5] where galoisian ideals are introduced.

The maximal ideal of $\underline{\alpha}$-relations $\mathfrak{M} = Id(\underline{\alpha})$ has as stabilizer

$$G = Stab_{\mathfrak{S}_n}(\mathfrak{M}) \ ,$$

which is the *Galois group* of $\underline{\alpha}$ in k.

By the natural k-morphism $x_i \to \alpha_i$ from $k[x_1, \ldots, x_n]$ to $k[\alpha_1, \ldots, \alpha_n] = k(\alpha_1, \ldots, \alpha_n)$, the field $k(\alpha_1, \ldots, \alpha_n)$ of the roots of f is isomorphic to the quotient ring $k[x_1, \ldots, x_n]/\mathfrak{M}$. The goal of the constructive Galois theory is to construct $\mathfrak{M}$ and to determine the Galois group G. One of the methods, called `GaloisIdeal` algorithm(see [5]) is based on a construction of an ascending chain

$$I_1 \subset I_2 \subset \cdots \subset \mathfrak{M}$$

of particular ideals called *galoisian ideals*, defined below. For the first ideal I_1, it is always possible to take the ideal $Id(S_n.\underline{\alpha})$, called the *ideal of symmetric relations*, which is generated by the Cauchy moduli, a triangular Groebner basis obtained by divided differences from the polynomial f. The resolvents have a double interest: construct a generator of I_{i+1} from I_i and simultaneously exclude some groups to be the Galois group of $\underline{\alpha}$ by using the *matrices of*

groups. More generally, the resolvents are intensively used in numeric and algebraic methods for computing the Galois group alone.

Let us define galoisian ideals and their injectors. Let L be a set of permutations of $\mathfrak{S}_n$ such that $L = GL$ (i.e. $G < L$ when L is a group). The ideal I of the variety $L.\underline{\alpha}$ is called a *galoisian ideal*, L is its *injector* in the galoisian ideal $\mathfrak{M}$; the algebraic variety of I is $V(I) = L.\underline{\alpha}$. Note that G is the injector of $\mathfrak{M}$ in itself and $\mathfrak{M} = Id(G.\underline{\alpha}) = Id(\underline{\alpha})$.

When the injector L of I in $\mathfrak{M}$ is a group, the galoisian ideal I is said *pure*. A galoisian ideal is pure if and only if L equals the stabilizer of I in $\mathfrak{S}_n$; it is itself equivalent to the inclusion of the Galois group G in this stabilizer. When I is pure $V(I) = L.\underline{\beta}$ for each $\underline{\beta} \in V(I)$. It is proved in [6] that a pure galoisian ideal is generated by a separable triangular set of polynomials; such an ideal is said *triangular*.

Definition 1 *The L-relative resolvent of $\underline{\alpha}$ by $\Theta \in k[x_1, \ldots, x_n]$ is the polynomial*

$$R_{\Theta,L,\underline{\alpha}} = \prod_{\Psi \in L.\Theta} (x - \Psi(\underline{\alpha})) \quad .$$

When I is pure, the resolvent does not depend on the choice of $\underline{\alpha}$ in $V(I)$; it thus can be denoted by $R_{\Theta,I}$.

The characteristic polynomial of the multiplicative endomorphism $\widehat{\Theta}$ induced by Θ in $k[x_1, \ldots, x_n]/I$ is a power of the resolvent :

$$\chi_{\widehat{\Theta},I} = R_{\Theta,I}^{\mathrm{card}(H)} \tag{1}$$

where $H < L$ is the stabilizer of Θ in L (Θ is called an *L-relative H-invariant*). By linear algebra $\chi_{\widehat{\Theta},I}$ belongs to $k[x]$. As the field k is perfect, $R_{\Theta,I}$ lies also in $k[x]$; moreover, if it is square-free then $R_{\Theta,I}$ is the minimal polynomial of $\widehat{\Theta}$, the square-free form of $\chi_{\widehat{\Theta},I}$.

In next section, we will apply Sentence 2 below to a subgroup K of L in order to compute L-relative resolvents. For this reason, we prefer to use respectively K and $J = Id(K.\underline{\alpha})$ instead of L and $I = Id(L.\underline{\alpha})$ in the rest of the present section.

Let $K < \mathfrak{S}_n$ and $\tau \in \mathfrak{S}_n$. Then we have

$$Id(K^{\tau^{-1}}.(\tau.\underline{\alpha})) = \tau^{-1}.Id(K.\underline{\alpha}) \quad . \tag{2}$$

Indeed, for each $\tau \in \mathfrak{S}_n$:

$$Id(\tau^{-1}K\tau.(\tau.\underline{\alpha})) = Id(K\tau.\underline{\alpha}) = \tau^{-1}.Id(K.\underline{\alpha}) \quad .$$

Sentence 1 *Let $K < \mathfrak{S}_n$, $J = Id(K.\underline{\alpha})$ and $\tau \in \mathfrak{S}_n$. The galoisian ideal $\tau^{-1}.J$ is pure with stabilizer $K^{\tau^{-1}}$ if and only if $G < K$, where G is the Galois group of $\underline{\alpha}$.*

Proof 1 *The Galois group of $\tau.\underline{\alpha}$ is the conjugate $G^{\tau^{-1}}$ of G, and the condition $G^{\tau^{-1}} < K^{\tau^{-1}}$ is equivalent to $G < K$. From Identity (2), the group $K^{\tau^{-1}}$ and $\tau.\underline{\alpha}$ define the galoisian ideal $\tau^{-1}.J$.*
As $K^{\tau^{-1}}$ is a group, the galoisian ideal $\tau^{-1}.J$ is pure with stabilizer $K^{\tau^{-1}}$ if and only if the Galois group of $\tau.\underline{\alpha}$ is a subgroup of $K^{\tau^{-1}}$ (see [5]); this is equivalent to $G < K$.

Sentence 2 *Let* $K < \mathfrak{S}_n$, $J = Id(K.\underline{\alpha})$ *and* $\tau \in \mathfrak{S}_n$. *Assume that the Galois group* G *of* $\underline{\alpha}$ *is a subgroup of* K. *Then*

$$R_{\Theta,\tau^{-1}.J} = R_{\tau.\Theta,J} \quad .$$

Proof 2 *The characteristic polynomial of* $\widehat{\Theta}$ *in* $k[x_1,\ldots,x_n]/\tau^{-1}.J$ *is*

$$\chi_{\hat{\Theta},\tau^{-1}.J} = \prod_{\sigma \in K^{\tau^{-1}}} (x - \sigma.\Theta(\underline{\beta}))$$

for any $\underline{\beta} \in V(\tau^{-1}.J)$ *since, by Lemma 1, the galoisian ideal* $\tau^{-1}.J$ *is pure with stabilizer* $K^{\tau^{-1}}$. *Thus, for any* $\underline{\beta} \in V(\tau^{-1}.J)$

$$R_{\Theta,\tau^{-1}.J} = \prod_{\sigma \in K^{\tau^{-1}}/Stab_{K^{\tau^{-1}}}(\Theta)} (x - \sigma.\Theta(\underline{\beta})) \tag{3}$$

Moreover,

$$\begin{aligned} Stab_{K^{\tau^{-1}}}(\Theta) &= \{\sigma \in \tau^{-1}K\tau \mid \sigma.\Theta = \Theta\} \\ &= \tau^{-1}\{\rho \in K \mid \tau^{-1}\rho\tau.\Theta = \Theta\}\tau \\ &= \tau^{-1}\{\rho \in K \mid \rho.(\tau.\Theta) = \tau.\Theta\}\tau \\ &= Stab_K(\tau.\Theta)^{\tau^{-1}} \end{aligned} \tag{4}$$

We can choose $\underline{\beta} = \tau.\underline{\alpha} \in V(\tau^{-1}.J) = K^{\tau^{-1}}.(\tau.\underline{\alpha})$ *(see Identity (2)). With the notations* $\rho = \tau\sigma\tau^{-1}$ *and* $S = Stab_K(\tau.\Theta)$, *Identities (3) and (4) imply*

$$\begin{aligned} R_{\Theta,\tau^{-1}.J} &= \prod_{\rho \in K/S} (x - \tau^{-1}\rho\tau.\Theta(\tau.\underline{\alpha})) \\ &= \prod_{\rho \in K/S} (x - \tau\tau^{-1}\rho\tau.\Theta(\underline{\alpha})) \\ &= \prod_{\rho \in K/S} (x - \rho.(\tau.\Theta)(\underline{\alpha})) \\ &= R_{\tau.\Theta,J} \end{aligned} \tag{5}$$

Remark 1 *From Identities (3) and (5), the following equality can be deduced more generaly for any subgroup* K *of* $\mathfrak{S}_n$*:*

$$\prod_{\sigma \in K^{\tau^{-1}}/S^{\tau^{-1}}} (x - \sigma.\Theta(\underline{\beta})) = \prod_{\rho \in K/S} (x - \rho.(\tau.\Theta)(\underline{\alpha})) \tag{6}$$

for any $\underline{\beta} \in V(\tau^{-1}.J)$ *where* $\tau \in \mathfrak{S}_n$ *and* $S = Stab_K(\tau.\Theta)$.

3 Double classes and resolvents

We are interested in computing the resolvent $R_{\Theta,I}$, where I is a pure galoisian ideal of stabilizer L. We show how a resolvent can be factored relatively to a *double transversal*. This can lead to decomposing the resolvent into a product of resolvents with smaller degrees, in particular when the polynomial f is reducible.

Following the notations of previous section, $H < L$ is the stabilizer of Θ in L. Let K be another subgroup of L. The relation $\mathcal{R}_{K,H} = \mathcal{R}$ defined in L by

$$\sigma \mathcal{R} \tau \quad \text{if} \quad \sigma H \cap K\tau \neq \emptyset$$

is an equivalence relation. The class of σ is called a *double class of L modulo K and H* and satisfies the following proposition:

Sentence 3 *Let $\sigma, \tau \in L$. Then $\sigma \mathcal{R} \tau$ if and only if $\tau \in K\sigma H$.*

Let us assume that we know a *double transversal*

$$K\backslash L/H = \{\tau_1, \ldots, \tau_m\}$$

of L modulo K and H, that is a set of representants of the equivalence classes of $\mathcal{R}$. We thus have

$$L.\Theta = \bigcup_{i=1}^{m} K\tau_i H.\Theta = \bigcup_{i=1}^{m} K\tau_i.\Theta \quad .$$

In order to decompose each term of the above union, we introduce the subgroups

$$H_i := K \cap H^{\tau_i}$$

of K for $i \in [\![1, m]\!]$.

Lemma 1 *Let $\tau_i \in L$ and H_i as above. If Θ is an L-relative H-invariant then $\tau_i.\Theta$ is a K-relative H_i-invariant.*

Proof 3 *For each permutation σ of H_i, there exists σ' in H such that*

$$\sigma\tau_i.\Theta = \tau_i\sigma'\tau_i^{-1}\tau_i.\Theta = \tau_i.\Theta \quad . \tag{7}$$

Therefore $\tau_i.\Theta$ is invariant under the action of H_i.

Now let $\sigma \in K$ which leaves $\tau_i.\Theta$ invariant. We show that σ belongs to H_i. From $\sigma\tau_i.\Theta = \tau_i.\Theta$ we deduce that Θ is invariant under the action of $\tau_i^{-1}\sigma\tau_i$. Then $\tau_i^{-1}\sigma\tau_i \in H$, in other words $\sigma \in \tau_i H \tau_i^{-1}$.

From Identity (7), we find by decomposing K according to a left transversal K/H_i of K:

$$\begin{aligned} K\tau_i.\Theta &= (K/H_i)H_i\tau_i.\Theta \\ &= (K/H_i)\tau_i.\Theta \ . \end{aligned} \tag{8}$$

Lemma 1 ensures us that the set $(K/H_i)\tau_i.\Theta$ has the same cardinality as the set K/H_i (i.e. the produced polynomials are pairwise distinct). Indeed, if there exist two permutations σ and σ' of K such that $\sigma\tau_i.\Theta = \sigma'\tau_i.\Theta$ then $\sigma^{-1}\sigma'$ is in H_i because it leaves $\tau_i.\Theta$ invariant. Then σ and σ' belong to the same left classe of K modulo K_i. Finally, we can write

Sentence 4 *Let* $K\backslash L/H = \{\tau_1, \ldots, \tau_m\}$ *and* $H_i = K \cap \tau_i H \tau_i^{-1}$ *for* $i \in [\![1, m]\!]$ *; we have*

$$L.\Theta = \bigcup_{i=1}^{m} (K/H_i)\tau_i.\Theta \quad , \tag{9}$$

where the union is disjoint.

Proof 4 *Equality (9) follows from identities (7) and (8). Moreover, assume that there exists two permutations* σ *and* σ' *of* K *such that* $\sigma\tau_i.\Theta = \sigma'\tau_j.\Theta$ *. Then* $\sigma\tau_i \in \sigma'\tau_j H$ *and consequently* $\tau_i \in K\tau_j H$ *, that is contradictory with definition of the double transversal.*

Sentence 5 *Let* $H = Stab_L(\Theta)$ *,* $K\backslash L/H = \{\tau_1, \ldots, \tau_m\}$ *,* $K_i = K^{\tau_i^{-1}}$ *,* $H_i = K \cap H^{\tau_i}$ *and* $H_i' = K_i \cap H$ *for* $i \in [\![1, m]\!]$ *. Then*

$$R_{\Theta,I} = \prod_{i=1}^{m} \prod_{\sigma \in (K/H_i)} (x - \sigma\tau_i.\Theta(\underline{\alpha})) \tag{10}$$

$$= \prod_{i=1}^{m} \prod_{\sigma \in (K_i/H_i')} (x - \sigma.\Theta(\tau_i.\underline{\alpha})) \tag{11}$$

with $H_i = Stab_K(\tau_i.\Theta)$ *and* $H_i' = Stab_{K_i}(\Theta)$ *for* $i \in [\![1, m]\!]$ *.*

Proof 5 *From Proposition 4, we express the resolvent as follows:*

$$R_{\Theta,I} = \prod_{i=1}^{m} \prod_{\Psi \in (K/H_i)\tau_i.\Theta} (x - \Psi(\underline{\alpha}))$$

$$= \prod_{i=1}^{m} \prod_{\sigma \in (K/H_i)} (x - \sigma\tau_i.\Theta(\underline{\alpha}))$$

by Lemma 1. By the same lemma $H_i = Stab_K(\tau_i.\Theta)$ *and one can easily verifies that* $H_i' = Stab_{K_i}(\Theta)$ *. Then Equality (11) follows from Remark 1.*

The resolvent $R_{\Theta,I}$ is algebraically computable by the algorithms in [6] or [7] based on successive resultants when a triangular basis of I is given. Anyway their costs may be dramatically reduced if it is possible to split the computation in several resolvents relative to galoisian ideals with generators of smaller degrees. By the independance of these factors the computation becomes parallelisable. Following these considerations an effective decomposition of $R_{\Theta,I}$ is given below.

Theorem 1 *Let* $H = Stab_L(\Theta)$ *and* $K\backslash L/H = \{\tau_1, \ldots, \tau_m\}$ *. If* $G < K$ *and if a triangular basis of* $J = Id(K.\underline{\alpha})$ *is given, then for each* $i \in [\![1, m]\!]$ *the resolvent* $R_{\Theta,\tau_i^{-1}.J}$ *is computable and*

$$R_{\Theta,I} = \prod_{i=1}^{m} R_{\Theta,\tau_i^{-1}.J} = \prod_{i=1}^{m} R_{\tau_i.\Theta,J} \quad . \tag{12}$$

Proof 6 *Let* $J_i = \tau_i^{-1}.J$. *If* $G < K$ *then by Lemma 1, the ideals* J *and* J_i *are pure with respective stabilizers* K *and* $K^{\tau_i^{-1}}$. *Therefore Sentence 2 and Relations (10) and (11) leads to Identities (12). Furthermore* $R_{\Theta,\tau_i^{-1}.J}$ *is computable since it is expressed as a resolvent relative to* J.

4 Case of reducible polynomials and application to $\mathbb{F}_p[x]$

As the goal of this paper is to compute the resolvent $R_{\Theta,I}$ by multimodular techniques when the base field is $\mathbb{Q}$, we will apply Theorem 1 to $\mathbb{F}_p[x]$ for f reducible over $\mathbb{F}_p$. Consequently, in this section the polynomial f is supposed to be reducible over k. In order to split the computation of the resolvent $R_{\Theta,I}$ we intend to determine a subgroup K of L containing the Galois group G, and such that the triangular basis of the associated galoisian ideal $J = Id(K.\underline{\alpha})$ is quickly computable.

Let $f = f_1 \cdots f_r$, $f_i \in k[x]$. For each i in $[\![1, r]\!]$, we denote by d_i the degree of f_i and by G_i the Galois group (over k) of $\underline{\alpha}_i$, a d_i-tuple of the d_i roots of f_i.

It is well known that there exists a conjugate G^τ of the Galois group G, $\tau \in \mathfrak{S}_n$, such that $G^\tau < G_{1,\ldots,r} = G_1 \times \cdots \times G_r$. For the the goal of the paper, it is sufficient to consider the case where $G_{1,\ldots,r} < L^\tau$. For $\sigma = \tau^{-1}$, the group $G^\sigma_{1,\ldots,r}$ satisfies the following condition:

$$G < G^\sigma_{1,\ldots,r} < L. \tag{13}$$

We first show how a triangular basis of the ideal $I' = Id(G^\sigma_{1,\ldots,r}.\underline{\alpha})$ can be obtained. Let $\mathfrak{M}_1, \ldots, \mathfrak{M}_r$ be the r maximal galoisian ideals of the respective $\underline{\alpha}_i$-relations. For each $i \in [\![1, r]\!]$, we can rename the variables appearing in the triangular basis of $\mathfrak{M}_i$ as a tuple $\underline{y_i}$, and consider the ideal $\mathfrak{M}_i$ in the ring $k[\underline{y_i}]$. In this context, let us denote by $T_i(\underline{y_i})$ a triangular generating set of $\mathfrak{M}_i$.

Let $\mathcal{T}$ be the triangular set formed by the union of $T_1, \ldots, T_r$, and $\mathcal{T}'$ obtained by replacing in $\mathcal{T}$ each variable $y_{i,j}$ by a variable x_s such that this substitution is one-to-one (among the set of variables $y_{i,j}$ and the set of variables x_s) and such that the pure galoisian ideal I' generated by $\mathcal{T}'$ has $G^\sigma_{1,\ldots,r}$ as stabilizer.

Remark 2 *To obtain* $\mathcal{T}'$ *easily, we compute the triangular set* $\mathcal{T}''$ *resulting from the following substitution in* $\mathcal{T}$ *:*

$$y_{1,1} := x_1,\ y_{1,2} := x_2,\ \ldots\ ,\ y_{r,d_r} := x_n\ ;$$

the set T'' *is a triangular generating set of the galoisian ideal with* $G_{1,\ldots,r}$ *as stabilizer (for a good choice of the conjugate of each* G_i*). Therefore the known identities about galoisian ideals give us:*

$$\mathcal{T}' = \sigma^{-1}.\mathcal{T}'' \quad .$$

As $G < G^\sigma_{1,\ldots,r}$ any group K such that

$$G^\sigma_{1,\ldots,r} < K < L \tag{14}$$

is a candidate for our goals, and the galoisian correspondance about ideals implies:

$$\begin{aligned} I = Id(L.\underline{\alpha}) &\subset J = Id(K.\underline{\alpha}) \\ &\subset I' = Id(G^{\sigma}_{1,\ldots,r}.\underline{\alpha}) \\ &\subset \mathfrak{M} = Id(G.\underline{\alpha}) \quad . \end{aligned}$$

Example 1 *When* $K = \mathfrak{S}_{d_1} \times \ldots \times \mathfrak{S}_{d_r}$ *the triangular basis of the ideal* J *is the union of the triangular bases of the galoisian ideals of symmetric relations of the polynomials* $f_1, \ldots, f_r$, *given respectively by the Cauchy moduli of* f_i *(see [8]). If moreover* L *is the symmetric group then we are in the particular situation studied by N. Rennert in [4] in order to compute absolute resolvents ; in this case,* $\mathcal{T}' = \mathcal{T}''$ *(i.e.* $\sigma = \mathrm{id}$*) and the condition (13) is satisfied for* $\sigma = \mathrm{id}$.

To simplify the rest of this presentation, we assume without lost of generality that

$$G < G_{1,\ldots,r} < L$$

and that the n-tuple $\underline{\alpha}$ of roots of f in $V(I)$ is ordered as well: the d_i-tuple $\underline{\alpha}_i$ of roots of f_i stands after the roots of f_{i-1} and before the roots of f_{i+1}.

Let $U_1, \ldots, U_r$ be r groups such that $G_i < U_i < \mathfrak{S}_{d_i}$ for $i = 1, \ldots, r$ and $U_1 \times \ldots \times U_r < L$. We can set

$$K = U_1 \times \ldots \times U_r \quad .$$

The union of the triangular Groebner bases of the ideals $Id(U_i.\underline{\alpha}_i)$ forms a triangular basis of J. Remark that this property has been already applied in order to describe the construction of a triangular basis of the ideal I'.
Practically, we choose groups U_i as small as possible such that the computation of $R_{\Theta,K.\underline{\alpha}}$ is the fastest includind the cost of a triangular basis of $Id(U_i.\underline{\alpha}_i)$.

Application to $\mathbb{F}_p[x]$

The coefficients of f belongs to $\mathbb{F}_p$, where p is a prime integer. In this particular case, the respective Galois groups G_i of f_i are the cyclic groups C_{d_i} of degree d_i. Denote by $\mathfrak{M}_i$ the (maximal) galoisian ideal of $\underline{\alpha}_i$-relations. The variety of $\mathfrak{M}_i$ is $C_{d_i}.\underline{\alpha}_i$. As the Galois group is cyclic, the triangular basis of $\mathfrak{M}_i$ can be computed easily from the irreducible factors of f_i in $\mathbb{F}_p[x]/ < f_i >$. Note that it is not necessary to factorise f_i completely (see [3]). The best choice is $U_i = C_{d_i}$ for $i = 1, \ldots, r$. We have just to find $\sigma \in \mathfrak{S}_n$ such that

$$K = (C_{d_1} \times \cdots \times C_{d_r})^{\sigma} < L. \tag{15}$$

5 Computation by multimodular technique

Let $f \in \mathbb{Z}[x]$ be any polynomial of degree n with n distinct roots in $\mathbb{C}$. We want to compute $R = R_{\Theta,L,\underline{\alpha}}$ for a group L containing the Galois group G of $\underline{\alpha}$.

Suppose that we computed the resolvent R modulo prime numbers $p_1, \ldots, p_s$ such that the product $p_1 \cdots p_s$ is greater than the double of the maximal absolute value of the coefficients of R. Then the resolvent R is computable by the Chinese Remainder Theorem.

In this section, we have to compute efficiently R modulo some prime p and to establish a bound on the coefficients of R. In addition, we give a certification to stop the algorithm before the bound is reached.

Assume that the integer p does not divide the discriminant of f. Such an integer, called *unramified*, exists since f is square-free. Set $\hat{g} = g \mod p$ for any polynomial g. Recall the essential following theorems:

Theorem 2 *(Dedekind, [9]) Let $f(x) \in \mathbb{Z}[x]$ be a polynomial of degree n with n distinct roots in $\mathbb{C}$ and let G be the Galois group of f over $\mathbb{Q}$ in $\mathfrak{S}_n$ (i.e. for any $\underline{\alpha}$). If p is unramified and $\hat{f} = \hat{f}_1 \cdots \hat{f}_r$ with $\hat{f}_i$ irreducible over $\mathbb{F}_p$ of degree d_i, then there exists $\tau \in G$ with a cycle decomposition $\sigma_1 \ldots \sigma_r$ with σ_i of length d_i.*

The tuple $(d_1, \ldots, d_r)$ of Theorem 2 is called the *cycle pattern* of σ and *the decomposition type* of $\hat{f}$.

Theorem 3 *(Frobenius Density Theorem, [10]) Let $(d_1, \ldots, d_r)$ be a partition of n. Then, the relative density of the set of primes p for which f modulo p has a given decomposition type $(d_1, \ldots, d_r)$ exists and equals $1/|G|$ times the number of $\sigma \in G$ with cycle pattern $(d_1, \ldots, d_r)$.*

Note that Frobenius Density Theorem is extended by Tchebotarev Density Theorem [11].

5.1 Computing R modulo p

In $\mathbb{F}_p[x]$, the polynomial $\hat{f}$ factorises into r irreducible factors as follows:

$$\hat{f} = \hat{f}_1 \cdots \hat{f}_r$$

where $deg(\hat{f}_i) = d_i$. When $r = 1$ the prime integer p is "bad"' and we throw this integer. Frobenius Density Theorem 3 shows the density of "good" primes.

Let $\hat{G}_i = C_{d_i}$ be the Galois group over $\mathbb{F}_p[x]$ of a d_i-tuple $\underline{\hat{\alpha}}_i$ of roots of f_i, $i \in [\![1, r]\!]$, and $\hat{G}$ be the Galois group of $\underline{\hat{\alpha}}$ over $\mathbb{F}_p[x]$; $\underline{\hat{\alpha}}$ can be chosen such that $\hat{G} < \hat{G}_1 \times \cdots \times \hat{G}_r$.

As p is unramified, for some $\sigma \in \mathfrak{S}_n$, by Dedekind Theorem 2, this inclusion follows:

$$(\hat{G})^\sigma < (\hat{G}_1 \times \cdots \times \hat{G}_r)^\sigma < G < L \quad .$$

We are exactly in the situation in which the computation by decomposition of the L-relative resolvent of $\underline{\hat{\alpha}}$ can be performed efficiently (see Section 4).

5.2 Bounding the coefficients of the resolvent $R_{\Theta,I}$

For the general case of relative resolvents, we just have to modify the Rennert's formulae ([4]) by replacing the symmetric group $\mathfrak{S}_n$, stabilizing the ideal of symmetric relations, by the group L, stabilizing the galoisian ideal I. This leads to the following expression for a bound on the coefficients of f.

$$B(R) = (C_\Theta(C_f + 1))^{dD_\Theta}$$

where C_g is the largest coefficient of a polynomial g in $\mathbb{Z}[x_1, \ldots, x_n]$, D_g its total degree, and $d = \deg(R) = [L : H]$.

5.3 Efficient probabilistic solution with certification

Since the above bound may need approximatively $[L : H]D_\Theta$ prime integers to obtain the resolvent, a probabilistic approach is interesting to limit these necessary primes. Let us denote by

$$\hat{R}^q = R \mod q \quad .$$

When $q = p_1 \cdots p_j$, where $p_1 \ldots, p_j$ are prime numbers, the polynomial $\hat{R}^q$ can be lifted by the Chinese Remainder Algorithm from the polynomials $\hat{R}^{p_i}$. A classical way to obtain the resolvent with high probability consists in returning $\hat{R}^q$ as soon as $\hat{R}^q = \hat{R}^{q'}$ where $q' = p_1 \ldots, p_j, p_{j+1}$.

We actually use another test to stop the computation. In [12], the condition $R(\Theta) \in I$ is exploited as a certification for numerical computations of resolvents. Following this idea, even though q is smaller than $2B(R)$, we cut the computation when $\hat{R}^q = 0$ modulo I. As the ideal I is triangular, this test is reduced to only n euclidean divisions in $\mathbb{Q}[x_1, \ldots, x_n]$.

6 The parallel algorithm

Let p be a prime number not dividing the discriminant of f. We denote by K_p the subgroup of L chosen in order to construct the resolvent $\hat{R}^p$ by means of the double transversal $K_p \backslash L / H$. The cardinality of this double transversal will be denoted by m_p.

A total parallel computation of the multi-resolvent $\hat{R}^p$ would require $m_p + 1$ processors : a principal processor and m_p secondary processors to compute each resolvent in the product of Theorem 1. For a given p, following Theorem 1, each branch computes a resolvent $R_{\tau_i.\Theta,J}$. In practice, these computations of resolvents take always similar times. It seems impossible to characterize a different behaviour since the timings depend essentially on the respective stabilisators of $\tau_i.\Theta$, which are pairwise conjugate groups. For instance, the method of [6] computes the resolvent by successive resultants of the polynomial $(x - \tau_i.\Theta)$ with respect to the triangular Groebner basis of J ; there is no reason that this sequence of resultants leads to significant different timings as we compute a resolvent by $\tau_i.\Theta$ or by $\tau_j.\Theta$. However, we cannot assure rigorously that the degree of parallelism is $m_p + 1$.

The difficulty of this algorithm is closely related with the two levels of parallelisation of the method. When the lifting of the resolvent by Chinese remainder is based on s prime integers $p_1, \ldots, p_s$ to obtain a certified result, more than $m_{p_1} + \ldots + m_{p_s}$ processors are required for a total parallelisation. In practice, this generally leads to a partial parallelisation, and it is not easy to decide and handle the repartition of the processors with respect to the two different tasks. Since a solution may be lifted with high probability without the computation of every $\hat{R}^{p_1}, \ldots, \hat{R}^{p_s}$, we naturally privilege the total computation of some resolvents $\hat{R}^p$. We suppose that $S + 1$ processors proc(0), ... ,proc(S) are available for the computation. The execution is controled by the master processor proc(0).

Step 1 /* This step refers to Section 5 */
Processor proc(0)
1.1 Compute a list P of unramified prime integers $p_1, \ldots, p_s$ such that $p_1 \cdots p_s > 2B(R)$.
1.2 $S_0 := min\{s, S\}$.

Step 2 /* This step refers to Section 4 */
For each processor proc(i), i=1 to S_0, do
2.1 Compute $\hat{f}^{p_i}$
2.2 Factorise $\hat{f}^{p_i}$ in $\mathbb{F}_{p_i}[x]$ into irreducible factors
Let r be the number of factors.
2.3 Deduce $G_1, \ldots, G_r$ the respective Galois groups of each factor (the cyclic groups)
2.4 Compute K_{p_i} (see Condition (15))
2.5 Compute a double transversal $\mathcal{D}$ of $K_{p_i} \backslash L / H$ and $m_i := m_{p_i}$
2.6 For $j = 1, \ldots, r$ compute the maximal ideals $\mathfrak{M}_j$ (actually their triangular basis)
2.7 Compute the triangular basis of the ideal J associated to K_{p_i} from the ideals $\mathfrak{M}_j$
2.8 Send $p_i, K_{p_i}, \mathcal{D}, J, m_i$ to the principal processor proc(0)

Step 3
/* This step refers to Section 5 for proc(S) and to Section 3 for the others.
Note 1: proc(S) is kept free for the computation of R by Chinese Remainder Theorem
Note 2: We estimate that the modular resolvents will be computed in similar times. */

Processor proc(0)

▷ Receive the above respective data from proc(1) to proc(S_0) and stores them in a list ℓ
▷ While (ℓ is not empty) do
  – Compute the largest integer $u \in [\![1, S_0]\!]$ and the number S_1 of processors such that $S_1 = (m_1 + 1) + \ldots + (m_u + 1) < S$
  – Delegate the computation of the resolvents $\hat{R}^{p_i}$ ($i \in [\![1, u]\!]$), to the u processors proc(N_i) ($1 \leqslant i \leqslant u$), where $N_1 = 1$ and $N_i = m_1 + \cdots + m_{i-1} + i$ if $i > 1$
  – Receive a boolean from proc(S) in the variable STOP
  – If STOP=true Then Send a Signal to proc(i) ($1 \leqslant i \leqslant S - 1$) ; Break ; End If
  – Delete the u first elements of the list ℓ
  – End While
▷ If STOP=true Then Receive the resolvent R from proc(S) ; Return R ; End If
▷ Delete $p_1, \ldots, p_{S_0}$ in the list P / * see Step 1 */
▷ $s :=$ length (P)
▷ $S_0 := min\{s, S - 1\}$
▷ Return to Step 2

Processors proc(j), $1 \leqslant j \leqslant S_1$
/* When the S_1 processors needed to obtain the modular resolvents receive the Signal from proc(0) their computations are simultaneously stopped. */

▷ If $j \in \{N_1, \ldots, N_u\}$ Then
  – Distribute the computation on the m_i processors proc($N_i + 1$), ..., proc($N_{i+1} - 1$) by sending to them $p_i, \Theta, K_{p_i}, H, J$ and $\tau \in \mathcal{D}$, the double transversal with $\#\mathcal{D} = m_i$

– Wait to gather the results
– Compute the product of these results
– Send the product to proc(S)

▷ Else
– Receive $p_i, \Theta, K_{p_i}, H, J, \tau$ from some proc(N_i)
– Perform the computation of $R_{\Theta,\tau.J} \mod p_i$ mentioned in Theorem 1
– Send $R_{\Theta,\tau.J} \mod p_i$ to proc(N_i)

▷ End If

Processor proc(S)
/* *Note 1*: this is actually repeated until the boolean variable STOP is set to true, meaning that the algorithm lifted a value for R. As mentioned above, we reserved proc(S) for this task that may be performed independently. Remark that proc(S) could be replaced by a set of processors working by parallel to lift R.
Note 2: we introduce variables m and F that contain respectively the product of the primes already taken into account and the current value of $\hat{R}^m$. */

▷ STOP := false
▷ Send STOP to proc(0)
▷ While not STOP do
– Receive $p_1, \ldots, p_u$ and $\hat{R}^{p_1}, \ldots, \hat{R}^{p_u}$ from proc(0)
– Let $m := m\, p_1 \cdots p_u$
– Compute $\hat{R}^m$ from F and the $\hat{R}^{p_i}$ by Chinese Remainder Theorem
– $F := \hat{R}^m$
– If $m > 2B(R)$ or $F(\Theta) \in I$ Then
STOP := True
End If
– Send STOP to proc(0)
– End While

▷ Send F to proc(0).

Conclusion and further developments

A part of this paper has been employed to establish Theorem 1 and to solve the problems of its application. These problems did not appear in Rennert's paper for computing absolute resolvents (see Exemple 1). It is important to note that our method is also more efficient in his context ($L = \mathfrak{S}_n$). Indeed, the group K in the decomposition

$$R_{\Theta,I} = \prod_{i=1}^{m} R_{\tau_i.\Theta,J}$$

of the theorem where $J = Id(K.\underline{\alpha})$, may be much smaller than the product of symmetric groups involved in Rennert's paper. In particular, in our algorithm K is a product of cyclic groups in the computation of the modular resolvents. Furthermore, in Rennert's paper the parallel strategy of his algorithm is not described though his practical exemple is sufficient to illustrate the interest of the methodology.

The modular computations have a double interest implying the double parallel character of the method. Indeed, each modular resolvent is computed in parallel and Theorem 1 is applied to compute in parallel factors of each modular resolvent. This doubly parallel character makes the implementation rather technical. However it opens a field of investigations and development of efficient strategies on how to optimize the distribution of the work on the processors between the different parallelisations involved in Steps 2 and 3 of the algorithm.

The modular computation of a resolvent produces some useful informations :

- One can apply the standard technique to exclude some groups to be the Galois group because each factorisation of f mod p (p an unramified prime) gives a subgroup of the Galois group of f (see Theorem 2).
- A partial factorisation of the resolvent R^{p_i} on $\mathbb{F}_i[x]$ is a by-product of the algorithm; these factorisations of modular resolvents could be memorized in view of a future factorisation of the resolvent R on $\mathbb{Z}[x]$ that is useful for computing minimal polynomials of algebraic numbers, the Galois group or galoisian ideals.

In some recent works, the double classes of groups have been exploited to study galoisian ideals and resolvents. The theoritical results of the present paper show their importance. This tool should probably leads to some future developments and understanding in Galois theory.

References

1. *Lagrange J.* Réflexions sur la résolution algébrique des équations. Prussian Academy, 1770.
2. *Galois E.* Oeuvres Mathématiques, éditées par la SMF. Paris: Gauthier-Villars, 1897.
3. *Valibouze A.* Sur les relations entre les racines d'un polynôme//Acta Arithmetica. 2008. V. 131.1. P. 1-27.
4. *Rennert N.* A parallel multi-modular algorithm for computing Lagrange resolvents//J. Symb. Comput. 2004. V. 37. N. 5. P. 547-556.
5. *Valibouze A.* Étude des relations algébriques entre les racines d'un polynôme d'une variable//Bull. Belg. Math. Soc. Simon Stevin. 1999. V. 6. N. 4. P. 507-535.
6. *Aubry P., Valibouze A.* Using Galois ideals for computing relative resolvents//J. Symbolic Comput. 2000. V. 30. N. 6. P. 635-651.
7. *Aubry P., Valibouze A.* Calcul algébrique efficace de résolvantes relatives. Archives HAL-CNRS, 2009. URL: http://hal.archives-ouvertes.fr/hal-00406357/en/.

8. *Rennert N., Valibouze A.* Calcul de résolvantes avec les modules de Cauchy//Experiment. Math. 1999. V. 8. N. 4. P. 351-366.

9. *Dedekind R.* Sur la théorie des nombres entiers algébriques. Paris:Gauthier-Villars, 1877.

10. *Frobenius F.G.* Über beziehungen zwischen den primidealen eines algebraischen körpers und den substitutionen seiner gruppe//Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu Berlin. Phys.-math. 1896. P. 689-703.

11. *Chebotarëv N.G.* Opredelenie plotnosti sovokuponosti prostykh chisel, prnadlezhashchikh zadannomu klassu podstanovok (determination of the density of the set of prime numbers belonging to a given substitution class)//Izv. Ross. Akad. Nauk. 1923. V. 17. P. 205-250.

12. *Abdeljaouad I., Bouazizi F., Valibouze A.* Certification algébrique pour le calcul de la résolvante de Lagrange. Archives HAL-CNRS, 2010. URL: http://hal.archives-ouvertes.fr/hal-00483257/en/.

Accepted for edition 7.06.2010.

ПАРАЛЛЕЛЬНОЕ ВЫЧИСЛЕНИЕ РЕЗОЛЬВЕНТ ЛАГРАНЖА С ПОМОЩЬЮ МУЛЬТИРЕЗОЛЬВЕНТ

© **Филипп Обри**
Университет Пьера и Мари Кюри, Париж, 75252, Франция, доктор наук, профессор,
e-mail: Philippe.Aubry@upmc.fr

© **Анник Валибуз**
Университет Пьера и Мари Кюри, Париж, 75252, Франция, доктор наук, профессор,
e-mail: Annick.Valibouze@upmc.fr

Ключевые слова: резольвента Лагранжа; группа Галуа; идеал Галуа; компьютерная алгебра; параллельные вычисления.

Целью данной работы является создание параллельного алгоритма вычисления резольвенты Лагранжа для полинома одной переменной. Вычисление резольвенты Лагранжа для полинома одной переменной важно для теории Галуа. Начиная с алгоритма Лагранжа, было получено много других частных резольвент, называемых абсолютными, по основной теореме о симметрических функциях. Алгоритмов для не абсолютных резольвент мало и они получены недавно, так как они используют идеалы Галуа, которые были введены недавно. Эти алгоритмы с ростом степени полинома требуют больших затрат времени и памяти. Поэтому требуется распараллеливание. В 2004 году N. Rennert предложил модулярный алгоритм для вычисления абсолютных резольвент для целочисленных полиномов. Мы показываем, что его техника может быть применена для любых резольвент. Такой алгоритм естественно распараллеливается. Кроме того, мы предлагаем формулу для разложения резольвент, которая дает дополнительное распараллеливание. Тем самым мы получаем алгоритм с двумя уровнями распараллеливания.

UDC 519.612

SOME ESTIMATIONS OF PERFORMANCE OF PARALLEL ALGORITHMS FOR SOLVING LARGE LINEAR SYSTEMS OVER GF(2)

© **Mikhail Alekseevich Cherepniov**
Moscow State University named after M.V. Lomonosov, Leninskiye gory, 1, Moscow, 119899, Russia, Candidate of Physics and Mathematics, Associate Professor Numbers Theory Department, e-mail: cherepniov@gmail.com

Key words: fast algorithms, sparce linear systems, parallel algorithms, computer algebra.
This topic explains how to estimate the running time and RAM volume required by programs of Wiedemann-Coppersmith algorithm, Montgomery's algorithm, some modifications of them and new algorithm when uploading multiple compute nodes and some other details of these algorithms.

1 Introduction

In principle, each operation of any algorithm can be considered for possible use of several similar sites to reduce time of their implementation. However, if the number of arithmetic operations is fixed, the most significant reduced time - this is where time falls proportional to the growing of compute nodes' number. It will be that, for example, if you search relations in the methods of the discrete logarithm or factorization problem based on the factor databases. However, for most complex algorithms top rated time of their work $T(N, n, d, c \dots; s)$ depend on task and used equipment parameters: N, n, d, c and the number of used compute nodes s and may behave differently on different areas of argument changing. Often when you increase s above a certain threshold $s_0(N, n, d, c \dots)$, depending on the task settings, time not only falls, but begins to grow again. That is because time for exchange between s computing nodes grows faster than of running time on each compute node separately reduce.

According to the author it is legitimately to raise the issue of calculating of values $s_0(N, d, n, c \dots)$, and $T_0(N, d, n, c \dots) = \min_s T(N, d, n, c \dots; s) = T(N, d, n, c \dots; s_0(N, d, n, c \dots))$ for some model cluster. As a model it is logical today to take the cluster with an unlimited number of compute nodes for which runtime of one arithmetic operation with machine words, multiplied by the some constant c, equal to the runtime of one machine word's passing between nods.

In modern computer network runtime of passing between the processor and RAM memory have $c \approx 5$, and between the individual parts of RAM $c \approx 20$.

As known by author, option c is determined by the ratio between the frequencies of the processor and "bus which links compute nodes, portion of information bits transferred on the internal network communications and some other characteristics of the cluster. Let's also assume that on a cluster there exists a possibility of direct delivery between performing specific job computing nodes; and broadcast from fixed site to the sites of some group by binary tree, i.e. by the logarithm of the number of elements of this group transfers. In this article, all the time calculates in units equal to time per arithmetic operation with machine words in such model

cluster. We assume that time for conversion of matrix formats and delay time for preparing network is negligible.

The task will characterize by four parameters: $N \geqslant 2^{26}$ - size of the original matrix (maximum number of rows and columns), d - upper bound of the number of nonzero elements (units) in each row of this matrix, n - machine word length, c - parameter which was mentioned above. However, if the volume of carriage, in bits, is V, while sending time will compute by formula $\frac{c}{n}V$ units of time. One unit time is the time that takes one operation with machine words in our model cluster.

2 Notes about symmetric matrixes

Let $N, M \in \mathbb{N}, \mathbb{F} = GF(2)$ and we want to obtain solution of system of linear homogeneous equations

$$DX = 0, D \in \mathbb{F}^{M \times N}, X \in \mathbb{F}^{N \times n}, M < N, \tag{1}$$

where n – width of block, typically equals to machine word length (32 or 64).

At the beginning of Montgomery's algorithm, and new algorithm [1] select random block $Y \in \mathbb{F}^{N \times n}$ and consider linear system

$$AX = B, A \in \mathbb{F}^{N \times N}; B = AY, X \in \mathbb{F}^{N \times n}, \tag{2}$$

where $A = D^T D \in \mathbb{F}^{N \times N}$ is singular symmetric matrix. Note that any solution X_D of the system (1) allows you to build a X_A - solution of the system (2) by the formula

$$X_A = X_D + Y.$$

Back, if X_A - is a solution of system (2), than $D^T D(X_A - Y) = 0$. If $rangD = M$, it follows that $D(X_A - Y) = 0$. That is $X_A - Y$ - solution of the system (1).

Let $dim_1 X$ is a dimension of the intersection of space $\langle X \rangle$, formed by columns of block X and the kernel of a linear operator D. Let $dim_2 X$ is a dimension of the intersection of the space, formed by columns of block X and the kernel of a linear operator $A = D^T D$, and $dim'_2 X$ - dimension of the intersection of space $\langle X \rangle$ and the kernel of a linear operator $A' = DD^T$

Theorem 1 *For arbitrary* $X \in \mathbb{F}^{N \times n}$

1. $dim_1 X \geqslant dim_2 X - (M - rangD)$
2. $dim_1 D^T X \geqslant dim'_2 X - (M - rangD)$

Proof.

1. In accordance with the size of the matrix D we have $dimKerD^T = M - rangD$. By definition, the vector X is in $KerA$ if and only if it is in $KerD$ or $DX \in KerD^T \setminus \{0\}$. The maximum number of linearly independent vectors that meet the second of these conditions is not more than $dimKerD^T$. So corang $KerD$ in $KerA$ not more than $M - rangD$. Crossing with $\langle X \rangle$ we obtain inequality 1.

2. The number of linearly independent vectors among columns of block D^TX decrease compared to the number of linearly independent vectors among columns of block X no more than $dimKerD^T = M - rangD$. So corang $KerD^T$ in $KerA'$ no more than $M - rangD$. Crossing the $\langle X\rangle$ with $KerA'$, we get confirmation of 2.

The theorem is proved.

Note that according to the dimensions, matrix A in general have more linear independent vectors in the core than matrix A'. So we will prefer A.

Theorem 2 *Let Krylov space $\langle W\rangle$ built as a sum of $\langle W_i\rangle, i = 0, 1, \ldots, m$, with an initial unit type $W_0 = B = AY$, on the first $m-1$ of which A-scalar production is nonsingular. Let $\langle W_m\rangle$ - A-orthogonal to whole $\langle W\rangle$, in particular to itself. Let along with $AW_i, i = 0, 1, ..., m-1$, also calculated AW_m and X by the formula*

$$X = \sum_{i=0}^{m-1} W_i(W_i^TAW_i)^{-1}W_i^TB, \tag{3}$$

$dim\langle W_m\rangle = \mu > 0$. Then by not more than $(n+\mu-1)(n+\mu)N$ bitwise operations with probability at least $1-\frac{1}{2^n}$ (subject to statistical independence $\langle Y\rangle$ and $\langle W\rangle$) one can calculate the solution of system (2).

Proof.

Note, that by condition the space $\langle W_m\rangle$ A-orthogonal to the space $W = \langle W_0\rangle + \cdots + \langle W_m\rangle$, in particular A-orthogonal to itself. By the way, note that in practice dimension $\langle W_m\rangle$ is small (say 2). Also by the the condition $\langle W_m\rangle$ contains all vectors from Krylov space that are A-orthogonal to all this space. Then AW_m have the form $w_0 + w_1 + \cdots + w_m, w_i \subseteq \langle W_i\rangle, i = 0, 1, 2, \ldots, m$. When $j \in \{0, 1, \ldots, m-1\}$ we obtain a chain of equations

$$w_j{}^TAW_j = (w_0 + \cdots + w_m)^TAW_j = (AW_m)^TAW_j =$$

$$(W_m)^TA(AW_j) \subseteq (W_m)^TAW = \{0\}$$

that, due to non singularity of A-scalar product at W_j shows that $w_j = 0$ for $j = 0, 1, \ldots, m-1$, i.e. $AW_m \subseteq \langle W_m\rangle$.

If $\langle AW_m\rangle \neq \langle W_m\rangle$, the element from the kernel of the matrix A can be obtained by reduction of right part of equality $AW_m = Z$ to triangular form by $2\frac{(\mu-1)\mu}{2}N$ operations (The complexity of one elimination with columns of matrices $AW_m\|W_m, Z \in \mathbb{F}^{N\times(n+\mu)}$ estimated with value N).

Now let $\langle AW_m\rangle = \langle W_m\rangle$. From formula (3) we get by substitution

$$(W_j)^T(AX - B) = (W_j)^TB - (W_j)^TB = 0, j = 0, 1, ..., m-1,$$

$$(W_m)^T(AX - B) = (W_m)^TA(AX - B) = 0,$$

because $AX - B \subseteq W$. Therefore, in view of the A-invariance of Krylov space

$$(W_j)^TA(AX - B) = (AW_j)^T(AX - B) \subseteq$$
$$W^T(AX - B) = \{0\}, j = 0, 1, ..., m-1,$$

that is vector of the block $AX-B$ are A-orthogonal to W , i.e. $\langle AX-B\rangle \subseteq \langle W_m\rangle$. It means that linear operator A displays sum of spaces $\langle X-Y\rangle + \langle W_m\rangle$ in $\langle W_m\rangle$. If $\langle Y\rangle \not\subseteq \langle W\rangle$, dimension of the first space is larger than the dimension of the second, and with the assistance of the Gaussian exceptions in equality $A(X-Y\|W_m) = Z, \langle Z\rangle \subseteq \langle W_m\rangle$, item from the kernel of A can be found.

The probability that the $\langle Y\rangle \not\subseteq \langle W\rangle$ subject to the statistical independence of these spaces and singularity of matrix A , that leads to $\langle W\rangle \neq \mathbb{F}^N$, obviously estimated by value $1-\frac{1}{2^n}$, where n - the number of columns in Y . The number of eliminations with columns of matrices $X-Y\|W_m, Z \in \mathbb{F}^{N\times(n+\mu)}$ estimated with value $2((n+\mu-1)+...+1) = 2\frac{(n+\mu)(n+\mu-1)}{2}$. Theorem is proven.

Note that condition $W_0 = B$ in the last theorem may be replaced by $W_0 = Y$ without subject to statistical independence $\langle Y\rangle$ and $\langle W\rangle$. Then similarly we obtain $A(X-Y) \subseteq \langle W_m\rangle$, where by construction $\langle X-Y\rangle \subseteq \langle W\rangle \setminus \langle W_m\rangle$.

3 Parallelization of first and third phases

Before discussing the wording of theorems, refer that consistent implementations of considered algorithms picks up maximum time for repeatedly recurring operation of multiplying matrix and block of vectors, this is the main part of the first and third phases of algorithms. Therefore, parallelization must be primarily applied to this operation.

You can consider the two approaches to parallelization of multiplication matrix and block of vectors related to distributed storage of the matrix and (or) distributed storage of the block.

The need to use the first of these approaches is also the impossibility of storing in memory of one compute node all researched matrix D .

To use the second approach consider block of ns vectors. Namely, that all consider algorithms apply to a system of linear equations $DX = 0$, where $D \in \mathbb{F}^{M\times N}, X \in \mathbb{F}^{N\times ns}$. The amount s called block factor. Let for simplicity $M = N$.

Consider the task of parallel distributed multiplication of sparse matrix on block.

To ensure balance load its need to separate sparse matrix into parts, containing approximately the same number of units. Thus without additional transformation, matrix can be separated in one direction - by ratios. Divide matrix $D \in \mathbb{F}^{N\times N}$ of our system of linear homogeneous equations on horizontal strips by number l of used compute nodes. Thus processor with number i will receive matrix D_i , in which the nonzero left only N/l matrix rows of D .

In the first phase of Montgomery's and the new algorithm it is necessary to calculate $(D^TD)^iB$, $B^T(D^TD)^iB, B \in \mathbb{F}^{N\times ns}, i = 1, 2, \ldots$.

Have

$$D^TD = \sum_{i=1}^{l} D_i^TD_i,$$

$$D^TDB_j = \sum_{i=1}^{l} D_i^TD_iB_j, j = 1, \ldots, s, \tag{4}$$

where $B_j \in \mathbb{F}^{N\times n}$ - is j vertical strip of $B \in \mathbb{F}^{N\times ns}$. Storage of this strips and matrix D_i on the compute node need memory

$$Nn + \frac{nNd}{l}$$

bit (we assume that every non-zero matrix element D_i have the row number and column number in one machine word).

Thus distributed computation of block vector D^TDB on ls compute nodes will require $\frac{N}{l}d$ operations for calculate items D_iB_j and $\frac{Nd}{l}$ (number of nonzero elements in the matrix ${D_i}^T$) operations to complete the calculation $D_i^TD_iB_j \in \mathbb{F}^{N\times n}$.

Further, the algorithm consistently evaluated $(D^TD)^kB$. This requires sending with addition by formula (4) and inverse mailing received left part of this equality for all l using in its calculation computer nodes of computing site. Using the cyclic sending this takes time $\frac{c}{n}2Nn = 2Nc$.

To sum up, we get a time to calculate $(D^TD)B$ at ls compute nodes in the form

$$\frac{2Nd}{l} + 2Nc. \tag{5}$$

Calculation of the total Krylov space needed in the new algorithm and in algorithm of Wiedemann-Coppersmith, to build the solution. In addition to the first phase of the algorithms vectors that constituents Krylov space uses for constructing coefficients of series as scalar products X^TD^iY for algorithm of Wiedemann-Coppersmith, and $B^T(D^TD)^iB$ for the new algorithm.

For distributed by rows (strips) storage at l compute nodes total block B^T even need $\frac{Nns}{l}$ bits. After computing the $(D^TD)^iB_j$ on each of the l compute nodes of group with number $j, 1 \leqslant j \leqslant s$, it calculates its part of scalar production $B^T(D^TD)^iB_j$ by

$$3\frac{Nns}{l(log_2\frac{N}{2} - log_2log_2\frac{N}{2})}$$

arithmetic operations (for corresponding algorithm one can see pp. 342-343 [4]). When $N \geqslant 2^{26}$ this fraction can be evaluated from the top by value

$$\frac{4Nns}{l \cdot log_2N}. \tag{6}$$

All parts after concatenation inside one group gives $B^T(D^TD)^iB_j$, and for all groups - $B^T(D^TD)^iB$. Sending time is the minor due to small size of sent matrices. Adding to the evaluation (5) we obtain estimate of calculation time for the next block of vectors from Krylov space and the coefficient of the series on the cluster with sl computing nodes:

$$\frac{2dN}{l} + \frac{4Nns}{l \cdot log_2N} + 2Nc. \tag{7}$$

To simplify the calculations we will use the value of l, that makes equal resulting components:

$$l = \frac{d}{c}. \tag{8}$$

A more detailed calculation shows that the second element in the estimation (7) when considered in the present values of parameters, namely, $2ns < dlog_2N$ less than the third. For

large values of s it's better to do scalar multiplications on the additional $O(\frac{s}{log_2 N})$ compute nodes so that the second element of the evaluation (7) be less than the first and the third. So let's use estimation

$$4Nc \tag{9}$$

with overall estimation throughout the first phase of the $2\frac{N}{ns}$ steps approximately

$$\frac{8N^2c}{ns} \tag{10}$$

with condition $sl \leqslant C$ or $s\frac{d}{c} \leqslant C$ where C - general number of compute nodes of the cluster.

The memory of one node that participate in multiplication matrix on the block and calculating coefficients of the series, should have RAM space about

$$\underset{\text{matrix } D_i;}{\frac{nNd}{l}} + \underset{\text{current } B_j \text{ ;}}{Nn} + \underset{\text{part of } B^T}{\frac{Nns}{l}} = \frac{Nn}{8\cdot 10^9}(1 + c + \frac{sc}{d})\text{GB}. \tag{11}$$

when $l = \frac{d}{c}$. In the case of $N \approx 2^{26}$ this is approximately

$$\frac{1}{2}(1 + c + \frac{sc}{d})\text{GB}. \tag{12}$$

Note that if scalar multiplications calculates on another additional compute nodes, then memory on a single node enough demand up

$$\frac{1+c}{2}\text{GB}.$$

For obtaining general evaluation of the first and the third phase of the new algorithm in various applications, estimation (10) must be multiplied by a constant, not a great 2.

In the case of Widemann-Coppersmith algorithm rating for memory usage gives expression (12). Estimate for running time may be obtain similar to (7), where there is no factor 2 neer d, as there is no D^T. So l will be determined by the formula

$$l = \frac{d}{2c}.$$

For the time of the first phase, contains the calculation of the $\frac{2N}{ns}$ series coefficients and taking into account the need to build two passages (second to build the solution), we get very close to the previous estimation:

$$\frac{16N^2c}{sn}. \tag{13}$$

Note only that in the second pass of algorithm Widemann-Coppersmith, operation similar to multiplication by B^T replaced by right multiplying on a relatively small matrices g_i that requires equivalent time.

4 Parallelization of the second phase

In the algorithm of Widemann in version of Coppersmith vector $\bar{g}$ builds consistently increasing its size. Giorgi P., Jannerod C-P., Villard G. in [3] proposed another option with almost linear estimation of complexity, though with several logarithmic and rather big absolute multiplicative constants. So optimized algorithm we will call algorithm of Widemann-Coppersmith with matrix polynomial multiplication.

Let

$$h = h(\lambda) = \sum_{i=0}^{\infty} H_i \lambda^i, H_i \in \mathbb{F}^{ns \times ns}. \tag{14}$$

Here the size of ns selected so that the algorithm can be apply to the construction of approximation polynomial at the second stage of the algorithm Widemann-Coppersmith using block factor s.

Definition 1 *Degree of vector polynomial* $m(\lambda) \in (\mathbb{F}[\lambda])^{2ns \times 1}$ *called its degree as a polynomial with vector coefficients from* $\mathbb{F}^{2ns \times 1}$.

Definition 2 *Order of vector polynomial* $m(\lambda) \in (F[\lambda])^{2ns \times 1}$ *is a nonnegative integer* j, *that satisfies* $h(\lambda)m(\lambda) = \sum_{i \geqslant j+1} \mu_i \lambda^i, \mu_{j+1} \neq \overline{0}$.

Definition 3 σ*-basis of the series* $h(\lambda)$ *let's call matrix polynomial* $M(\lambda) \in \mathbb{F}^{2ns \times 2ns}[\lambda]$ *that satisfies the following conditions:*

1. The columns $M^{(i)}(\lambda)$ *of matrix* $M(\lambda)$ *have an order of not less than* σ.

2. Any $v \in \mathbb{F}^{2ns \times 1}[\lambda]$, *whose order is not less than* σ, *have unique representation*

$$v = \sum_{i=1}^{2ns} M^{(i)} C^{(i)}, C^{(i)} \in \mathbb{F}[\lambda], degM^{(i)} + degC^{(i)} \leqslant degv.$$

To implement Widemann-Coppersmith algorithm, we must build Pade approximations to series of the form (14), where $H_i = X^T D^i Y; X, Y \in \mathbb{F}^{N \times ns}$, namely $P(\lambda), Q(\lambda) \in \mathbb{F}^{ns \times 2ns}[\lambda]$, that satisfies

$$(h \parallel I_{ns}) \begin{pmatrix} Q \\ P \end{pmatrix} = O(\lambda^{2d+1}), degQ, degP \leqslant d, \tag{15}$$

where I_{ns} in concatenation is identity matrix from $\mathbb{F}^{ns \times ns}, d = \frac{N}{ns}$.

Let further for simplicity $Nns = 2^k$.

Theorem 3 *Let* $N \geqslant 2^{26}$. *An upper bound for running time of parallel implementation using block factor* s *of the second phase of the Wiedemann-Coppersmith algorithm with matrix polynomial multiplication*

1) When

$$s \geqslant \frac{120(log_2 log_2 32N)(log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2})}{17n} \tag{16}$$

has the form :

$$1122Nn^2s^3(log_232N)(log_24N). \tag{17}$$

2) When

$$s \leqslant \frac{120(log_2log_232N)(log_2\frac{ns}{2} - log_2log_2\frac{ns}{2})}{17n} \tag{18}$$

has the form :

$$15840Nns^2(log_232N)(log_24N)(log_2log_232N). \tag{19}$$

Remark 1 *Because inequality* $z \geqslant 2Alog_2A$ *leads to* $z \geqslant Alog_2z$, *we obtain that if* $s \geqslant 2\frac{120log_2log_232N}{17n}log_2\left(\frac{120log_2log_232N}{34}\right)$, *than inequality (16) holds.*

Proof of 3.

To build the necessary approximation we use the algorithm from article [3], which builds σ-bases, consistently doubling σ. According to this article we can built whole basis, when replace series $h(\lambda)$ to $h(\lambda^{2ns}) \times (1, \lambda, \lambda^2, \ldots, \lambda^{2ns-1})^T$ and construct approximation to it which order is $2dns$, using $2d$ coefficients of series $h(\lambda)$. Estimation of running time of program of corresponding algorithm $C(ns, ns, 2dns)$ can be obtained from the proof of theorem 2.4 [3], replacing d to $2dns$, namely

$$C(ns, ns, 2dns) \leqslant 2C(ns, ns, dns) +$$
$$MM(ns, dns) + MM(ns, 2dns),$$

where $MM(a, b)$-complexity of multiplying the two matrix polynomials with degree b from $\mathbb{F}^{a\times a}[\lambda]$. Continuing similarly, obtain

$$\begin{gathered} C(ns, ns, 2dns) \leqslant \\ MM(ns, 2dns) + \sum_{i=1}^{log_2 2dns} MM(ns, 2^{-i}2dns)(2^i + 2^{i-1}) \\ \leqslant \frac{3}{2}\sum_{i=0}^{log_2 2dns} 2^i MM(ns, 2^{-i}2dns). \end{gathered} \tag{20}$$

Remember, that for simplicity we demand $Nns = 2^k$.

Optimized algorithm to multiply matrix polynomials may be taken from the article [5]. According to lemma 3.2 of this work obtain following estimation

$$MM(ns, 2^i) \leqslant \alpha_Q\frac{(ns)^2}{n} + \beta_Q\frac{(ns)^2\frac{ns}{n}}{log_2\frac{ns}{2} - log_2log_2\frac{ns}{2}},$$

where $Q : \varphi(r^Q) \geqslant 2(2^i + 1), H : 2^H + 2 \leqslant Q \leqslant 2^{H+1} + 1$, $\alpha_Q \leqslant r^Q 2^H((6r + 2)\mu_r(H + 1) + \frac{2\alpha_2}{r^2})$, $\beta_Q \leqslant r^Q 2^H \beta_2$, where as r any natural number other than 2 can be selected, μ_r-sum of modules of the coefficients in a cyclotomic polynomial with number r, α_2 and β_2 respectively is the number of additions and multiplications for multiplication polynomials with degree $\varphi(r^2) - 1$. It is known that when $r = 3$ one can choose β_2 equals to 17. Here we apply trivial algorithm for addition of matrixes, broken by rows in machine words with the length of n and optimized algorithm of multiplying such matrixes as set out above. When $r = 3$ we get $\mu_r = 3$. Q can be chosen the lowest such that $3^{Q-1} \geqslant 2^i + 1$ (see. p. 8 [3]), so $3^Q \leqslant 9(2^i + 1)$, and when $i \geqslant 1$ we obtain $3^Q \leqslant 2^{i+4}$, and because $3log_32 < 2$, we have:

$$MM(ns, 2^i) \leqslant \frac{(ns)^2}{n} 2^{i+4} log_3 2^{i+4} \cdot$$
$$\left(20 \cdot 3(log_2 log_3 2^{i+4} + 1) + \frac{2}{9}\alpha_2 + \frac{17ns}{log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2}}\right)$$
$$\leqslant \frac{(ns)^2}{n} \frac{2^{i+5}}{3}(i+4) \cdot$$
$$\left(60(log_2 \frac{2(i+4)}{3} + 1) + \frac{2}{9}\alpha_2 + \frac{17ns}{log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2}}\right)$$

Because inequality

$$\frac{2}{9}\alpha_2 + 60 log_2 \frac{4(i+4)}{3} \leqslant 120 log_2 log_2 32dns$$

for $i \leqslant log_2 2dns$ obviously done, subject estimation can be continued by value

$$ns^2 2dns 11(log_2 2dns + 4) \cdot \left(120 log_2 log_2 32dns + \frac{17ns}{log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2}}\right) \quad (21)$$

Thus

$$C(ns, ns, 2dns) \leqslant$$
$$\frac{3}{2} ns^2 22 \cdot 2dns(log_2 2dns + 4) log_2 4dns \cdot \left(120 log_2 log_2 32dns + \frac{17ns}{log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2}}\right) \leqslant$$
$$66 N ns^2 log_2 32N log_2 4N \cdot \left(120 log_2 log_2 32N + \frac{17ns}{log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2}}\right) \quad (22)$$

In the case 1) this inequality can be continued by value

$$\frac{132 N ns^2 (log_2 32N)(log_2 4N) \cdot 17ns}{log_2 \frac{ns}{2} - log_2 log_2 \frac{ns}{2}}.$$

Since in this case, $\frac{ns}{2} \geqslant 16$, this evaluation can be continued by value

$$1122 N ns^2 (log_2 32N)(log_2 4N).$$

Second case obviously follows from the evaluation (22). Theorem proven.

Theorem 4 *Running time estimation of parallel implementations using block factor of Wiedemann-Coppersmith algorithm with the matrix polynomial multiplication under condition that* $N \geqslant 2^{26}$ *and*

$$N > \frac{71(log_2 32N)(log_2 4N)}{cn} \cdot \left(\frac{240 log_2 log_2 32N log_2 (\frac{60}{17n} log_2 log_2 32N)}{17}\right)^4$$

has the form

$$94N^{1+\frac{3}{4}}n^{-\frac{1}{4}}c^{\frac{3}{4}}((log_2 32N)(log_2 4N))^{\frac{1}{4}}.$$

Proof.
Proof of this theorem follows from remark 1, theorem 3, 1) and evaluation (13) when

$$s = \frac{1}{4}\left(\frac{8Nc}{561n^3(log_2 32N)(log_2 4N)}\right)^{\frac{1}{4}}.$$

Note that when $N \geqslant 2^{26}, c \geqslant 1$ estimation of this theorem holds.

5 Conclusion

It's worth noting that in procedures (see [6]) using which 12 December 2009 was received new record of integer factorization, actually applies the usual algorithm of Widemann in Coppersmith version with $s = 8$, but construction of approximations was not consistent, but by binary tree as described above. Complexity of the corresponding algorithm proposed by Thom$\acute{e}$ in the [7] estimates by value

$$O(n^2s^2(ns + log_2 k)k \cdot log_2 k \cdot log_2 log_2 k), k = \frac{N}{ns},$$

that is,

$$O(Nns(ns + log_2 N)log_2 N log_2 log_2 N).$$

This estimate is very close to evaluation (22). In principle, the difference is that degree of s is one less. Therefore, similar reasoning, when $s = O(N^{\frac{1}{3}})$ we can obtain time estimation of the type

$$O(N^{1+\frac{2}{3}}(log_2 N log_2 log_2 N)^{\frac{1}{3}}).$$

It is important to note that the necessary memory volume in this case

$$O(k(ns)^2 log_2 k) = O(Nnslog_2\frac{2N}{ns})$$

bits, where multiplier $log_2 k$ associated with increasing integer factors when the recursive application of Fourier transform for polynomials with integer coefficients (see, for example [8, 9]) is used. For values of parameters for which was done record calculations $N \approx 3 \cdot 2^{26}, s = 8$, it's around 1TB. With using optimal value of s this is $O(N^{1+\frac{1}{3}}nlog_2 N)$ that is really significantly much. An important advantage of algorithm [1] is the lack of need for multiplication of matrix polynomials to calculate only few coefficients such works. That leads to the good parallelization and runtime approximation for the second phase about $O(N)$ with the same asymptotic for RAM volume (with constant in O equals approximately 3).

However, some optimizations of this algorithm allows quite remove requirement growing RAM, and get an estimation of time of parallel implementations using block factor when $N \geqslant 2^{26}, n = 64, 1 \leqslant c$ of the whole algorithm of the type:

$$8,4c^{\frac{2}{3}}N^{1+\frac{2}{3}} \tag{23}$$

in assessing the number of compute nodes asymptotically $O(\frac{N^{\frac{2}{3}}}{logN})$ (when $N \approx 2^{26}, c \approx 12, d \approx 2^9$ number of required compute nodes is approximately 2400).

It is known that in usual parallel implementation of Montgomery's algorithm [2] the main complexity gives sending time during multiplying a matrix by a vector (see estimation (5)). You can assess that value by

$$\frac{N}{n}(\frac{c}{n}2Nn) = \frac{2cN^2}{n}.$$

As is known for the author, now there exist parallel implementations of Montgomery's algorithm with better bounds. Attitude this time work to the optimal time for the new algorithm roughly equal to 3. Increasing the size of the matrix N the attitude will be equal to

$$3\sqrt[3]{\frac{N}{2^{26}}}.$$

In addition, this algorithm have not "high parallelism" , i.e. its parts cannot be done on independent clusters. The new algorithm and algorithm of Wiedemann-Coppersmith allow you to compute the coefficients of series and its approximations on the independent sites.

Note ones more feature of Widemann-Coppersmith algorithm and the new algorithm. When you use block factor s , running time of the first and third phases can be reduced to s , running time of the second phase increases depending on s not slower than s^2 . For example, the complexity of multiplying two matrices from $GF(2)$ with size $ns \times ns$, as we saw above, have the estimate n^2s^3 . At the same time, if this matrix divide to the blocks with the size $ns^{\frac{2}{3}} \times ns^{\frac{2}{3}}$ and send each pair of such blocks from different matrices for multiplication on an execution site, while time on sending and arithmetic operations will be about $O(n^2s^2)$, and the number of used compute nodes $O(s^{\frac{4}{3}})$. Similar considerations can be done fore the parallelization of scalar production of vectors. Solving of homogeneous systems (bring to the triangular form) can be made using a recursive algorithm that transfers calculations again to a matrix multiplication. Such parallelization leads to overall estimate of the form

$$O(\frac{N^2}{s}) + O(Ns),$$

that for $s = O(\sqrt{N})$ leads to $O(N^{\frac{3}{2}})$. The total number of compute nodes $2sl$ (see. (8)) will be about $O(N/logN)$.

We gather results in the table:

	Running time	**Nodes**	**Total RAM**
Wiedemann-Coppersmith with matrix polynomial multiplication	$94N^{1+\frac{3}{4}}n^{-\frac{1}{4}}c^{\frac{3}{4}}((log_2 32N)(log_2 4N))^{\frac{1}{4}}$	$O\left(\frac{N^{\frac{1}{2}}}{(log_2 N)^2}\right)$	$O(N^{1+\frac{1}{4}})$
Wiedemann-Coppersmith-Thomé	$O(N^{1+\frac{2}{3}}(log_2 N log_2 log_2 N)^{\frac{1}{3}})$	$O(\frac{N^{\frac{2}{3}}}{log_2 N})$	$O(N^{1+\frac{1}{3}}log_2 N)$
Montgomery (1995)	$\frac{2c}{n}N^2$	$O(1)$	$O(N)$
New algorithm	$8,4c^{\frac{2}{3}}N^{1+\frac{2}{3}}$	$O(\frac{N^{\frac{2}{3}}}{log_2 N})$	$O(N^{1+\frac{1}{3}})$

References

1. *Cherepnev M.A.* Block Lanczos-type algorithm for solving sparse linear systems //Diskr. Math. (in Russian). 2008. V. 20. N. 1. P. 145-150.

2. *Montgomery P.L.* A Block Lanczos Algorithm for Finding Dependencies over GF(2)//EUROCRYPT'95, LNCS, 1995. V. 921. P. 106-120.

3. *Giorgi P., Jannerod C-P., Villard G.* On Complexity of Polynomial Matrix Computations//ISSAC'03, August 3-6, 2003. Philadelphia, USA.

4. *Coppersmith D.* Solving Homogeneous Linear Equations Over GF(2) via Block Wiedemann Algorithm//Mathematics of Computation. Jan. 1994. V. 62. N. 205. P. 333-350.

5. *Cantor D., Kaltofen E.* On Fast Multiplication of Polynomials Over Arbitrary Algebras//Acta Informatica.1991. V. 28. P. 693-701.

6. *Kleinjung T., Lenstra A.K., and others* Factorization of a 768-bit RSA modulus. version 1.0. January 7, 2010. URL: http://eprint.iacr.org/2010/006.pdf.

7. *Thomé E.* Subquadratic computation of vector generating polynomials and improvement of the block Wiedemann algorithm//J. of Symbolic Computation. 2002. V. 33. Issue 5. P. 757-775.

8. *Vasilenko O.N.* Number-Theoretic Algorithms in Cryptography //AMS. Moscow State University, 2007.

9. *Naudin P., Quitte C.* Algorritmique Algebrique. Masson, 1992.

Accepted for publication 7.06.2010.

НЕКОТОРЫЕ ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ РЕШЕНИЯ БОЛЬШИХ ЛИНЕЙНЫХ СИСТЕМ НАД $GF(2)$

© **Михаил Алексеевич Черепнев**
Московский государственный университет им. М.В. Ломоносова, Ленинские горы, 1, Москва, 119899, Россия, кандидат физико-математических наук, доцент кафедры теории чисел, e-mail: cherepniov@gmail.com

Ключевые слова: разреженные системы; факторизация целых чисел; параллельные алгоритмы; компьютерная алгебра.

В данной работе даны оценки времени, памяти при оптимальном числе узлов для известных блочных алгоритмов решения разреженных систем над $GF(2)$, их модификаций и нового алгоритма.

UDK 519.688

DFT FOR POLYNOMIALS IN PARALLEL ALGORITHMS

© **Aleksey Olegovich Lapaev**
Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Post-graduate Student of Computer and Mathematical Modeling Department, e-mail: alapaev@gmail.com

Key words: polynomials; discrete Fourier transform; parallel algorithm; method of homomorphic images; cluster.

We investigate sequential and parallel algorithms for polynomial arithmetic based on discrete Fourier transform (DFT). Algorithms for polynomial multiplication are discussed. Sequential algorithms for polynomial matrix are proposed. Each algorithm based on DFT has been compared with similar algorithm based on Chinese remainder theorem. In the last part of work parallel algorithms for calculation DFT and multivariable polynomials multiplication are considered. Theoretical expressions of complexity are presented for each algorithm. Results of experiments on MVS cluster are presented for parallel algorithms.

1 Introduction

Effective implementation of arithmetics for multivariable polynomials is a significant problem in symbolic computations. This problem is important for calculations with polynomial matrices. During calculations polynomials of high degrees are appearing. Therefore, standard algorithmes are non-effective. Modular methods are used to reduce cost of polynomial coefficient and degree growth. Let's remind Chinese remainder theorem (CRT). It is usually used by next scheme: elements of polynomial matrices are mapped into finite fields $Z_p[x]/(x-j)Z_p[x]$, $Z_p = Z/pZ$, where p is some prime number. Then calculations are fulfilled over this finite fields. Result is recovered via Newton's or Lagrange's scheme. Complexity of interpolating of one polynomial is $O(t^2 + tr^2)$, where t and r are the numbers of used polynomial and numerical modules respectively.

It has been shown in works [10, 11] that algorithms based on discrete Fourier transform are more effective at interpolating result instead of CRT for polynomial modules. The main idea of calculations based on DFT is that under mapping in factor ring $Z_p[x]/(x-j)Z_p[x]$ values of parameter j are chosen in a special way. It allows to recover result in $Z_p[x]$ by DFT and CRT in time $O(t\log_2 t + tr^2)$ where t – the number of points in DFT, r – the number of the numerical modules. Coefficients of polynomials are also recovered by CRT.

In paper [1] the algorithm for multiplication of two polynomials based on DFT in a finite field $Z_p[x]$ is described. Theoretical and experimental comparison of DFT-algorithm of polynomials multiplication with Karatsuba's algorithm of multiplication [12] and direct algorithm is resulted in works [7,12]. The problem of multiplication of two polynomials based on DFT on the processor with several kernels and the general memory is considered in articles [2,3].

Comparison of two classes algorithms for polynomial calculations is done in the given work: the first class of algorithms uses CRT both for polynomial and for the numerical modules. The second class of algorithms uses DFT instead of CRT for polynomial modules.

In section 2 theoretical and experimental comparison of algorithms of polynomial multiplication of one variable is spent.

In section 3 DFT-algorithm for calculation of a determinant, the characteristic polynomial, the adjoint matrix for the matrices which elements are polynomials of one variable over a ring of integers are considered. The problem of multiplication of two polynomial matrices is considered. For each algorithm theoretical and experimental comparison with the CRT-algorithm is resulted. Alogrithm based on DFT are suggested by author.

In section 4 the algorithm of calculation of DFT for multivariable polynomials is considered, complexity estimations are shown. The parallel algorithm of multidimensional DFT calculation is proposed by author. Results of experiments on cluster MVS are presented.

In section 5 the parallel algorithm of multivariable polynomial multiplication based on DFT on parallel machines with the distributed memory is considered. Parallezation scheme of algorithm has been made by the authors.

All experiments are done on MVS cluster with the next configuration: 1460 computing modules with 8 cores Intel Xeon 3 GHz and 8Gb RAM by module, operation system is Cent OS. All algorithmes were implemented in Java 1.6. MPIJava binding for MPICH is used for parallel algorithms.

2 Algorithms of multiplication of polynomials of one variable in Z-ring

It has been shown In work [1,7,12] that multiplication of polynomials based on DFT has the complexity estimation $O(m \log_2 m)$, where m is the degree of polynomials. Let's consider the following algorithm for computing product of two polynomials $f, g \in Z[x]$ based on DFT (PF):

1. Choose the number of numerical prime modules r, $p_0, \ldots, p_{r-1}$, sufficient for recovering of fg in Z.
2. Calculate DFT $F(f), F(g)$ for polynomials f and g in each finite field Z_{p_i}, $i = 0, \ldots, r - 1$.
3. Calculate $F(f) \cdot F(g)$ in each finite field $Z_{p_i}, ß = 0, \ldots, r-1$, where " $\cdot$ " - operation of element-wise multiplication of two vectors.
4. Calculate the inverse DFT for vector $F(f) \cdot F(g)$ in each finite field $Z_{p_i}, ß = 0, \ldots, r-1$. Elements of this vector are polynomial's coefficients fg over the module p_i
5. Reconstruct polynomial coefficients fg in Z by CRT.

At the given algorithm following mappings in various algebras take place:

$$Z[x] \to Z_p[x] \to^F F(Z_p[x]) \to^{F^{-1}} Z_p[x] \to Z[x]$$

Let's receive theoretical expressions for complexity of last algorithm. Let f and g be a polynomials in $Z[x]$ with degrees $m-1$, each coefficient occupies w machine words. Product degree fg is $2m-2$. The maximum by absolute value coefficient of product of polynomials fg contain no more than $r = \lceil \log_h m + 2w \rceil$ words, where $h = 2^H$, H - number of bits in a machine word.

Let's choose as modules prime numbers, each consists of H bits.

Then r is the number of prime modules $p_0, \ldots, p_{r-1}$ which is enough for recovering the result from $Z_{p_i}[x]$ to $Z[x]$ via modular a method according to the Chinese remainder theorem (CRT). Here it is designated $Z_{p_i} = Z/p_i Z$.

As product degree fg is equal $2m-2$, the number of points for DFT is $N = 2^{\lceil \log_2(2m-1) \rceil}$.

Let $\hat{f} = F_{p_i}^N(f)$ and $\hat{F}_{p_i}^N(\hat{f})$ - N-dimensional vectors of direct and inverse discrete Fourier transforms for a polynomial f in the field Z_{p_i} on N points. Here f is considered as a vector of polynomial's coefficients. It is possible to show that for any f it is carried out

$$Nf = \hat{F}_{p_i}^N(F_{p_i}^N(f)),$$

Then product of polynomials f and g in the field Z_{p_i} is calculated by the formula

$$\hat{F}_{p_i}^N(F_{p_i}^N(f) \cdot F_{p_i}^N(g))/N, \qquad (*)$$

where operation " $\cdot$ " designates multiplication of two vectors of length N in terms $V_i = W_i \cdot U_i$.

Let's result separately each step of algorithm and number of operations on the step.

a) We calculate remainders of division polynomials f and g by p_i, $i = 0, \ldots, r-1$. It takes to execute $2m$ divisions of numbers of length (w) on numbers of length (1). It is required $2mrw$ divisions and $2mrw$ subtractions.

b) We calculate DFT for polynomials f and g in $Z_{p_i}[x]$
$i = 0, \ldots, r-1$ Calculation of DFT is carried out by algorithm Cooley-Tukey [7] on $N = 2^{\lceil \log_2(2m-1) \rceil}$ points. Then the number of operations of addition, multiplication and division are the same and equal to $2Nr \log_2 N$.
c) Element-wise multiplication of DFT-images of polynomials f and g in each field Z_{p_i}. Thus it is required rN operations of multiplication and as much division operations.
d) We calculate the inverse Fourier transform in each field Z_{p_i}. For this purpose it is required to perform $rN \log_2 N$ operations of addition, multiplication and division.
e) We recover the result coefficients by CRT. For this recovery it is required to fulfill $2r^2(2m-1)$ addition and multiplication operations.

Let's compare the considered algorithm and the following:
0. Standard algorithm of multiplication for numbers and polynomials (PSS).
1. Karatsuba's algorithm for multiplication of numbers and standard algorithm for multiplication of polynomials (PSK).
2. Standard algorithm for multiplication of numbers and Karatsuba's algorithm for multiplication of polynomials (PKS).
3. Karatsuba's algorithm for multiplication both numbers and polynomials (PKK).

Let's result theoretical estimations of algorithms [12]. We get the number of operations of addition - $\mathcal{A}$, multiplication - $\mathcal{M}$ and divisions - $\mathcal{D}$.

Table 1

Number of additions, multiplications and divisions at multiplication of dense polynomials for algorithms 0-4

0 PSS	$\mathcal{A}$	m^2w^2
	$\mathcal{M}$	$m^2(w^2+2w)$
1 PSK	$\mathcal{A}$	$10m^2(w^{\log_2 3}-w)$
	$\mathcal{M}$	$m^2w^{\log_2 3}$
2 PKS	$\mathcal{A}$	$w^2(10m^{\log_2 3}-14m+4)$
	$\mathcal{M}$	$m^{\log_2 3}w^2$
3 PKK	$\mathcal{A}$	$w(10m^{\log_2 3}-14m+4)+(mw)^{\log_2 3}$
	$\mathcal{M}$	$(mw)^{\log_2 3}$
4 PF	$\mathcal{A}$	$2mrw+3Nr\log_2 N+2r^2(2m-1)$
	$\mathcal{M}$	$3Nr\log_2 N+rN+2r^2(2m-1)$
	$\mathcal{D}$	$2mrw+3rN\log_2 N+rN$

It is clear from Table 1 that the best algorithm by degree of polynomials is algorithm PF. The best algorithm by the number of words w in coefficients of polynomials are the algorithms based on Karatsuba's scheme for number multiplication.

2.1 Experimental comparison of algorithms

Programs have been written and experiments for measuring time of execution for algorithms of polynomial multiplication with corresponding parameters m and w are done.

Results are presented in tables below.

It is possible to choose for each set of parameters m and w algorithm which spends the least time for multiplication of two polynomials by comparison the tables listed above. Besides, let's find the relation of the time spent with algorithm PSS, to epy time spent with the most fast algorithm. Results are presented in Table 3.

Table 2

Results of experiments with multiplication of polynomials of one variable in a ring of integers

m	4	16	64	256	1024	4096	16384
PSS							
$w = 4$	0.008	0.235	5.9	103	1550	20930	292020
$w = 16$	0.05	1.325	26.5	425	6720	101970	$1.58{\cdot}10^6$
$w = 64$	0.68	17.25	305	5060	80740	$1.3{\cdot}10^6$	$21{\cdot}10^6$
PSK							
$w = 4$	0.009	0.26	6.4	109	1550	21770	319440
$w = 16$	0.05	1.3	25	415	6480	99700	$1{,}6{\cdot}10^6$
$w = 64$	0.66	19.75	330	5430	78730	1213330	$19.3{\cdot}10^6$
PKS							
$w = 4$	0.022	0.39	5	55	300	3600	28240
$w = 16$	0.057	0.85	10.1	102	870	6940	59040
$w = 64$	0.56	6.8	68	590	5360	47420	42679
PKK							
$w = 4$	0.023	0.4	5.2	57	510	3690	29830
$w = 16$	0.057	0.84	10.1	104	880	6990	59670
$w = 64$	0.55	11.2	90	720	5680	48040	513630
PF							
$w = 4$	0.33	1.575	7.8	34.5	145	670	2510
$w = 16$	1.9	9.6	41	175	730	2910	12200
$w = 64$	20.	88	360	1450	5890	24130	100140

Table 3

Numbers of algorithms of multiplication of dense polynomials which have shown in experiments the least time for the data m and w and their gain in time in relation to standard algorithm

m	$w = 4$	$w = 16$	$w = 64$
4	0/1	0/1	1/1.03
16	0/1	2/1.53	2/2.53
64	2/1.18	2/2.48	2/4.85
256	4/2.99	2/4.06	2/8.58
1024	4/10.69	4/8.87	2/14.69
4096	4/31.24	4/35.04	4/53.24
16384	4/116.34	4/129.94	4/204.84

The table 3 can be compared with theoretical estimations of complexity resulted in Table 1. For this purpose we construct Table 4 on expressions of complexity for the same sets of parameters m and w.

Table 4

Numbers of the best algorithms by theory for multiplication of dense polynomials and their prize concerning standard algorithm

m	$w = 4$	$w = 16$	$w = 64$
4	2/1.5	2/1.7	3/1.9
16	2/2.1	2/2.7	3/3.4
64	2/3.3	2/4.7	3/5.9
256	2/5.7	2/8.2	3/10.4
1024	2/10.0	2/14.5	4/20.4
4096	2/17.6	4/43.9	4/78.2
16384	4/61.8	4/160.4	4/300.0

From comparison of these two tables it is clear that theoretical expressions for complexity of the algorithms well enough correspond with times are received during experiments. Average relative error makes 35.86 %.

Distinctions in a prize are connected by that theoretical estimations consider only operations of addition, multiplication and division, laying aside all other operations.

3 DFT in consecutive matrix algorithms

3.1 Matrix multiplication

Let's consider a problem of multiplication of two matrices over a ring of polynomials of one variable. Let's result two algorithms of matrix multiplication over a ring of polynomials of one variable and receive expressions of their complexity:

0. Modular algorithm of matrix multiplication, using CRT both for polynomials, and for their coefficients (MCC).

1. Modular algorithm of matrix multiplication, using fast Fourier transform for polynomials and CRT for their coefficients (MFC).

In algorithm MFC the new way of application of discrete transformation of Fourier is used. For each element of multiplied matrices it is calculated homomorphic DFT-image, and further calculations on algorithm are made with these images. The result is recovered at first by inverse DFT and then by CRT after the end of calculations.

Let matrices $A, B \in M_{n\times n}[\mathbf{Z}[x]]$. Let maximum on the coefficients absolute value of polynomials which are elements of matrices A and B, contain w_A and w_B words accordingly. The maximum degrees of elements of matrices A and B it is less m_A and m_B. We get $m = \min\{m_A, m_B\}$. Then the maximum coefficient of product contains $r = \lceil w_A + w_B + \log_h m + \log_h n \rceil$ words, $h = 2^H$. Having assumed that in CRT for numbers are used H-bit prime modules, r is an enough number of modules, sufficient for result recovering.

The maximum degree of polynomials which appear in product $A \cdot B$ is less then $m_{AB} = m_A + m_B - 1$.

Let's take polynomial modules of the first degree. In algorithm MCC it is necessary to take m_{AB} prime polynomial modules $x, x-1, x-2, \ldots, x-m_{AB}+1$. In algorithm MFC it is required to calculate DFT for each element of matrices A and B on $N = = 2^{\lceil \log_2 m_{AB} \rceil}$ points. It is known from work [7] that with use of N points, DFT in a prime field can be calculated for $N \log_2 N$ operations of addition and as much multiplication and division operations.

At reflection in the field $Z_{p_i}[x]$ remainders of division by prime numbers $p_0, \ldots, p_{r-1}$ are calculated. Thus, it is required to execute $n^2 r m_A w_A$ and $n^2 r m_B w_B$ divisions and as much subtractions for matrices A and B accordingly. For refelection to homomorphic images in $Z_{p_i}[x]_j = Z_{p_i}[x] \diagup (x-j) Z_{p_i}[x], æ = 0.m_{AB} - 1$, it is necessary to calculate a polynomial remainders of division on the prime modules $(x-j)$ in the field $Z_{p_i}[x]$. For this purpose it is required $n^2 r m_{AB} m_A$ and $n^2 r m_{AB} m_B$ operations of multiplication, addition and division for matrices A and B accordingly.

For multiplication of matrices we use standard algorithm. Then the number of ring operations in $Z[x]$ is n^3 operations of multiplication and n^3 addition operations.

Complexity of addition of two polynomials is $r m_{AB}$ additions and $r m_{AB}$ divisions at calculations on algorithm MCC . Complexity of multiplication of two polynomials makes $r m_{AB}$ multiplications and as much divisions.

Complexity of addition of two polynomials is rN additions and as much divisions at calculations on algorithm MFC . Complexity of multiplication of two polynomials makes rN multiplications and rN divisions.

It is necessary to recover n^2 polynomials. It is required $m_{AB}^2 r$ multiplications and twice as much additions for recovery of one polynomial from $Z_{p_i}[x]_j$ to $Z_{p_i}[x]$. As addition and multiplication are carried out by the prime module p_i it is required also $3m_{AB}^2$ divisions. It is required $r^2 m_{AB}$ multiplications and twice more additions for restoration of factors of a polynomial in $Z[x]$.

Let's result the number of additions $\mathcal{A}$, multiplications $\mathcal{M}$ and divisions $\mathcal{D}$ for algorithms MCC and MFC.

Table 5

The functions are expressing the number of operations of addition, multiplication and division for algorithms MCC and MFC

0	$\mathcal{A}$	$n^2 r(m_A w_A + m_B w_B + m_{AB} m_A + + m_{AB} m_B) + n^3 r m_{AB} + + n^2(2m_{AB}^2 r + 2r^2 m_{AB})$
MCC	$\mathcal{M}$	$n^2 r(m_{AB} m_A + m_{AB} m_B) + n^3 r m_{AB} + + n^2(2m_{AB}^2 r + r^2 m_{AB})$
	$\mathcal{D}$	$n^2 r(m_A w_A + m_B w_B + m_{AB} m_A + + m_{AB} m_B) + 2n^3 r m_{AB} + 3n^2 r^2 m_{AB}$
1	$\mathcal{A}$	$n^2 r(m_A w_A + m_B w_B + 2N \log_2 N) + + n^3 r N + n^2(rN \log_2 N + 2r^2 m_{AB})$
MFC	$\mathcal{M}$	$2n^2 r N \log_2 N + n^3 r N + + n^2(rN \log_2 N + 2r^2 m_{AB})$
	$\mathcal{D}$	$n^2 r(m_A w_A + m_B w_B + 2N \log_2 N) + + 2n^3 r N + n^2 r N \log_2 N$

3.2 Experimental comparison of algorithms MCC and MFC

Programs have been written and experiments in which and MFC time of performance of algorithms MCC was measured in milliseconds for parameters m, n, w are made.

Let's consider that the matrix has the size n, density α which elements are degree polynomials m with the factors consisting from w of machine words, has type (n, m, α, w).

Let's result the relation of time of performance of algorithm MCC to time of performance of algorithm MFC:

Table 6

The relation of time of performance of algorithm MCC to algorithm MFC at multiplication of matrices of type $(n, m, 100, w)$ by results of experimental comparison

w=8					
m	n=4	n=8	n=16	n=32	n=64
4	2.01	1.77	1.6	1.23	1.04
8	3.49	3.05	2.22	1.48	1.15
16	5.61	3.99	2.99	1.84	–
32	7.71	5.85	3.43	1.82	–
64	11.74	6.65	–	–	–
128	13.91	7.95	–	–	–
w=16					
m	n=4	n=8	n=16	n=32	n=64
4	2.51	2.31	2.0	1.71	–
8	3.99	3.49	2.86	2.42	–
16	5.73	4.63	3.78	3.04	–
32	8.01	6.03	4.57	–	–

For some m, n, w values of the functions expressing complexity of algorithms MCC and MFC (Tab. 5) have been calculated. The theoretical prize of algorithm MFC rather MCC is presented to Table 7.

Table 7

The relation of the number of arithmetic operations of algorithm MCC to the number of arithmetic operations of algorithm MFC for type matrixes $(n, m, 100, w)$ by results of theoretical comparison

$w = 8$					
m	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
4	2.33	2.14	1.87	1.58	1.32
8	2.52	2.33	2.06	1.75	1.46
16	2.94	2.73	2.41	2.03	1.66
32	3.84	3.55	3.12	2.58	2.05
64	5.63	5.2	4.53	3.68	2.8
128	9.09	8.38	7.27	5.83	4.3
$w = 16$					
m	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$
4	2.81	2.63	2.35	1.99	1.63
8	2.94	2.77	2.5	2.14	1.77
16	3.23	3.05	2.76	2.37	1.95
32	3.88	3.66	3.31	2.82	2.28

By comparison of two last tables it is clear that theoretical and experimental estimations well correspond to themselves. The average relative distinction is equal to 24.37 %.

3.3 Algorithms of calculation of a determinant, a characteristic polynomial and the adjoint matrix

Let's receive theoretical expressions of complexity for algorithms of calculation of a determinant, a characteristic polynomial and the adjoint matrix for a matrix over polynomials of one variable.

Let $A = (a_{ij}(x))$ be a matrix over a ring $\mathbf{Z}[x]$ the size $n \times n$, $a_{ij}(x) = \sum\limits_{k=0}^{s_{ij}-1} a_{ij}^k x^k$. Let $\max\limits_{i,j,k} |a_{ij}^k| \leqslant \alpha$ and $\deg a_{ij} < s$.

Complexity of calculation of discrete Fourier transform on algorithm for n^2 polynomials on $N = 2^{\lceil \log_2 ns \rceil}$ points at use r modules is equal

$$n^2 r(7s\lceil \log_h \alpha \rceil + 9N \log_2 N).$$

3.3.1 Determinant calculation

Let's apply algorithm of a forward stroke to calculation of a determinant of a matrix. Let's estimate the number of prime 32-bit modules r, sufficient for determinant reception on its image at discrete Fourier transform. The matrix determinant $\det A$ can be calculated by the formula:

$$\det A = \sum_{(j_1,\ldots,j_n)} (-1)^t a_{1j_1} a_{2j_2} \ldots a_{nj_n}, \tag{1}$$

where $(j_1, \ldots, j_n)$ - transposition of numbers from 1 to n, t - signum of this transposition.

The formula (1) contains exactly $n!$ composes. Having estimated the maximum coefficeint of $\det A$ and having taken advantage of the Stirling's formula for the top estimation of value $n!$, we receive that the number of prime modules r be equal

$$r = \lceil \log_h 2(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}} s^{n-1} \alpha^n) \rceil,$$

where $h = 2^{32}$.

Each ring operation over images of polynomials consists from rN arithmetic operations over words and rN calculations of a remainder of division. We consider that calculation of a remainder of division occupies as much time, as 7 additions of words. Then the number of arithmetic operations over images of polynomials of an matrix A using the algorithm of a forward stroke equally

$$8n^3 rN.$$

For polynomial recovering from it's image it is necessary to execute r inverse DFTs and interpolate ns numbers. Thus, the number of operations for polynomial recovering from it's image DFT be

$$9rN \log_2 N + 2r^2 sn.$$

The total of operations with use DFT in algorithm of calculation of a determinant is equal

$$n^2 r(7s\lceil \log_h \alpha \rceil + 9N \log_2 N) + 8n^3 rN + 9rN \log_2 N + 2r^2 sn. \tag{2}$$

Let's result expression of complexity of algorithm of calculation of the determinant, using CRT both for polynomials, and for their factors:

$$n^2 r(7s\lceil \log_h \alpha \rceil + ns^2) + 8n^4 sr + 2n^2 s^2 r + 2r^2 sn. \tag{3}$$

$N = 2^{\lceil \log_2 ns \rceil}$ in (2), (3).

If ns is equally exact degree of number 2 estimations (2) and (3) look likes (4) and (5):

$$n^2 r(7s\lceil \log_h \alpha \rceil + 9ns \log_2(ns)) + 8n^4 rs + 9ns \log_2(ns) \cdot r + 2r^2 sn, \tag{4}$$

$$n^2 r(7s\lceil \log_h \alpha \rceil + ns^2) + 8n^4 rs + 2n^2 s^2 \cdot r + 2r^2 sn. \tag{5}$$

3.3.2 Calculation of a characteristic polynomial

The algorithm for calculation of characteristic polynomial of a matrix [6] is known. The given algorithm also has complexity $O(n^3)$ by ring operations. The difference is in a value estimation r and it is necessary to recover n polynomials. For the given algorithm $r_1 = \log_h 2n^n s^{n-1} \alpha^n$. Complexity of this algorithm is

$$n^2 r_1(7s\lceil \log_h \alpha \rceil + 9N \log_2 N) + 8n^3 r_1 N + n(9r_1 N \log_2 N + 2r_1^2 sn). \tag{6}$$

The classical approach with use CRT both for polynomials, and for numbers has complexity

$$n^2 r_1(7s\lceil \log_h \alpha \rceil + ns^2) + 8n^4 r_1 s + n(2n^2 s^2 r_1 + 2r_1^2 sn). \tag{7}$$

If ns equally exact degree of number 2 estimations (6) and (7) look like (8) and (9):

$$n^2 r_1(7s\lceil \log_h \alpha \rceil + 9ns \log_2(ns)) + 8n^4 r_1 s + n(9ns \log_2(ns) \cdot r + 2r_1^2 sn), \tag{8}$$

$$n^2 r_1(7s\lceil \log_h \alpha \rceil + ns^2) + 8n^4 r_1 s + n(2n^2 s^2 \cdot r_1 + 2r_1^2 sn). \tag{9}$$

3.3.3 Calculation of the adjoint matrix

The algorithm of calculation of the adjoint matrix also has complexity $O(n^3)$ by ring operations. The number of prime modules is

$$r = \log_h 2(\sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\frac{1}{12n}} s^{n-1} \alpha^n).$$

It is necessary to recover n^2 polynomials from their images. Thus, the general complexity of algorithm is

$$n^2 r(7s\lceil \log_h \alpha \rceil + 9N \log_2 N) + 8n^3 rN + n^2(9rN \log_2 N + 2r^2 sn). \tag{10}$$

We also result the expression of complexity of a classical method:

$$n^2 r(7s\lceil \log_h \alpha \rceil + ns^2) + 8n^4 rs + n^2(2n^2 s^2 r + 2r^2 sn). \tag{11}$$

If ns is equally exact degree of number 2 estimations (10) and (11) looks like (12) and (13):

$$n^2 r(7s\lceil \log_h \alpha \rceil + 9ns \log_2(ns)) + 8n^4 rs + n^2(9ns \log_2(ns) \cdot r + 2r^2 sn), \tag{12}$$

$$n^2 r(7s\lceil \log_h \alpha \rceil + ns^2) + 8n^4 rs + n^2(2n^2 s^2 \cdot r + 2r^2 sn). \tag{13}$$

3.4 Experimental comparison between the algorithms of calculation of a determinant, a characteristic polynomial and the matrix

On expressions of complexity algorithms with use DFT are close to what use CRT for polynomials and CRT for coefficients. The difference only that complexity of calculation of remainders of division of polynomials on prime modules $x, x+1, \ldots, x+ns$ makes ns^2 and polynomial recovering from its remainders of division by prime modules demands $2n^2s^2$ operations, and computing the image of a polynomial at DFT and recovering a polynomial from its DFT-image demands $9N \log_2 N$ operations. As $N \sim ns$ DFT has advantage. By theoretical estimations the prize is greatest in case that ns is exact degree of number 2.

Let's present results of computing experiments.

Comparison of algorithms for calculation of a determinant, characteristic polynomial and the adjoint matrix was spent.

We compared two algorithms for calculation of a determinant: the first one is based on CRT only and the second one is based on FFT for polynomials.

For calculation of a characteristic polynomial of a matrix over a ring $Z[x]$ two algorithms were compared: Danilevsky's algorithm with application CRT for polynomial prime modules and New algorithm [6] in which operations over polynomials are replaced by operations over their images of transformation of Fourier. For restoration of factors of a polynomial as a result in both algorithms CRT was used.

For calculation of the ajoint matrix it was spent comparison of two algorithms: the first one is based on CRT only and the second one is based on FFT for polynomials.

Results are presented in Tables 8-10.

Table 8

Results of experiments with calculation of a determinant of a matrix

$s=2$, $bits=8$, n increases			
n	**CRT, ms**	**FFT, ms**	
2	38	26	1.46/1
4	30	13	2.3/1
8	70	48	1.45/1
16	1094	952	1.14/1
32	25009	22203	1.12/1
64	625352	506382	1.35/1

$n=8$, $bits=8$, s increases			
s	**CRT, ms**	**FFT, ms**	
2	76	51	1.48/1
4	132	124	1.06/1
8	276	249	1.11/1
16	755	501	1.51/1
32	1884	1010	1.87/1
64	5040	2538	1.99/1

Table 9

Results of experiments with calculation of a characteristic polynomial

$bits=32$, $s=8$, n increases			
n	**Danilevsky's algorithm, ms**	**New algorithm with DFT, ms**	
4	63	23	2.74/1
8	415	143	2.90/1
16	7655	3619	2.12/1
32	162543.0	99562	1.63/1

$bits = 32$, $n = 8$, s increases			
n	**Danilevsky's algorithm, ms**	**New algorithm with DFT, ms**	
16	1705	761	2.24/1
32	5586	1181	4.73/1
64	23204	2671	8.69/1

Table 10

Results of experiments with calculation of the adjoint matrix

$s = 2$, $bits = 8$, n increases			
n	**CRT,ms**	**FFT,ms**	
2	1	1	1/1
4	7	5	1.4/1
8	118	67	1.76/1
16	2574	1164	2.21/1
32	65679	27226	2.41/1
64	1775930	611129	2.91/1

$n = 8$, $bits = 8$, s increases			
s	**CRT,ms**	**FFT,ms**	
2	132	73	1.81/1
4	325	138	2.35/1
8	968	284	3.41/1
16	4034	589	6.85/1
32	14834	1171	12.67/1
64	55809	2354	23.71/1

It is clear from Tab. 8-10 that the greatest advantage with DFT has the approach in algorithm of calculation of the adjoint matrix.

4 Parallel algorithm of DFT calculation

Let $f \in \mathbb{Z}_p[x_1, x_2, \ldots, x_d]$, p - prime number. Let the greatest degree of a variable x_i is equal $n_i - 1$ in the polynomial f, $n_i = 2^{N_i}$. We designate that $n = n_1 n_2 \ldots n_d$. The polynomial f can be written down in a form:

$$f = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \ldots \sum_{i_d=0}^{n_d-1} f_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_d^{i_d}.$$

Let the prime number p be so that n_i divides $p-1$. Then in $\mathbb{Z}_p$ there is the root degree n_i from 1 which we designate ω_i. We enter definition of discrete Fourier transform for the polynomial f.

D e f i n i t i o n: Discrete transformation of Fourier (DFT) for a polynomial f is called $dthe$-dimensional table of numbers $\mathcal{F}(f) = (\hat{f}_{j_1 \ldots j_d})$, where $1 \leqslant j_1 \leqslant n_1$, $1 \leqslant j_2 \leqslant n_2$, $\ldots 1 \leqslant j_d \leqslant n_d$, where

$$\hat{f}_{j_1 j_2 \ldots j_d} = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \ldots \sum_{i_d=0}^{n_d-1} f_{i_1 i_2 \ldots i_n} \omega_1^{j_1 i_1} \omega_2^{j_2 i_2} \ldots \omega_d^{j_d i_d}. \tag{14}$$

The right part (14) contains n composes. In the left part there is an element $dthe$-dimensional table, the number of all elements is equal n.

Hence, total number of operations for calculation DFT of a polynomial f is $O(n^2)$. We consider a way of fast calculation of DFT. Let's write down the formula (1) in a kind:

$$\hat{f}_{j_1 j_2 \ldots j_d} = \sum_{i_d=0}^{n_d-1} \omega_d^{j_d i_d} \sum_{i_{d-1}=0}^{n_{d-1}-1} \omega_{d-1}^{j_{d-1} i_{d-1}} \ldots \sum_{i_k=0}^{n_k-1} \omega_k^{j_k i_k} \ldots \sum_{i_1=0}^{n_1-1} f_{i_1 i_2 \ldots i_d} \omega_1^{j_1 i_1}.$$

At the fixed parameters $i_2, \ldots, i_d$ expression

$$F^1_{j_1,i_2,i_3,\ldots,i_d} = \sum_{i_1=0}^{n_1-1} f_{i_1 i_2 \ldots i_d} \omega_d^{j_1 i_1} -$$

element with number j_1 of one-dimensional discrete transformation of Fourier on n_1 points which can be counted on algorithm Cooley-Tukey [1] for $O(n_1 \log_2 n_1)$ operations.

Let's designate

$$F^{k+1}_{j_1,\ldots,j_{k+1},i_{k+2},\ldots,i_d} = \sum_{i_{k+1}=0}^{n_{k+1}-1} F^k_{j_1,\ldots,j_k,i_{k+1}\ldots i_d} \omega_d^{j_{k+1} i_{k+1}}.$$

At consecutive calculation $F_1, F_2, \ldots, F_d$ the element F_d contains DFT a polynomial f. The parallelization scheme consists of d consecutive steps, each of them is carried out in parallel. It is possible to present the scheme of calculation of each step in the form of a binary tree at which the data is distributed from root vertex to leaf, and the result of calculations gathers again in the root. On each step in parallel to be calculated one-dimensional DFT, thus all calculations occur in leaf vertexes. At transition to a following step the order of variables varies, and after a step d the required d-dimensional vector of DFT is received. We are resulting the algorithm.

Algorithm.

Let's spend calculations in a following order: on the step k are in parallel calculated n/n_k DFTs at the fixed values of indexes $j_1 \ldots j_{d-k-1} j_{d-k+1} \ldots j_d$.

Let F_k be the result of calculations of k-th step, and let $F_0 = f$.

Let's result the scheme of calculations at the step k. In root vertex d-dimensional array received on the previous step splits on two equal parts by an index j_k.

And each part goes to corresponding affiliated top. In affiliated tops division repeats by the same index on two half. Such process is carried out recursively to those while there are free processors or while division on the given index, i.e. number of the processors involved in calculations less n/n_k, is possible. At leaf level one-dimensional DFTs are calculated and the result comes back upside-down. Then, at the transition to a next step of calculations, the order of indexes of the array varies from $[j_{d-k+2} j_1 \ldots j_{d-k+1}]$ to $[j_{d-k+1} j_1 \ldots j_{d-k}]$. We notice that on the step k to it is required no more, than n/n_k processors.

Let there are m computer modules (CM) with numbers of $1 \ldots m$. Let $k = 1$.

1) The CM with number 1 receives d-dimensional array and the list of free CM. Degrees $w_{d-k+1}^{i_{d-k+1} j_{d-k+1}}$ are calculated. The array and the list of free processors are halved. One half of array and the list of free processors together with a array of degrees is sent to CM with number $(n/2+1)$, and second half remain in the given CM.

2) Each CM which has received the part of a problem, continues such division further. Process proceeds before achievement of sheet level, or exhaustion of all free processors.

3) Parallel calculation of one-dimensional Fourier transform is carried out.

4) Each CM sends result back on a tree to that processor from which has obtained the data.

5) The CM 1 collects result and changes an indexation order in result array from $[j_{d-k+2} j_1 j_{d-k+1}]$ to $[j_{d-k+1} j_1 \ldots j_{d-k}]$.

6) Number k increases. If $k = d+1$ the end of all calculations, differently we pass to item 1.

D e f i n i t i o n: Efficiency of computation on k processors with comparison on n processors is

$$a_{n,k} = \frac{\frac{t_n}{t_k} - 1}{\frac{k}{n} - 1}.$$

On this algorithm the program complex on cluster MVS the Russian Academy of Sciences is developed. Results are presented in Table 11.

Table 11

Time and speedup of parallel algorithm for calculation DFT for polynomial of two and three variables in finite field Z_p

d=2												
Time, ms							Efficiency, %					
n	512	1024	2048	4096	8192		n	512	1024	2048	4096	8192
procs							procs					
1	817	6838	47466	371882	67108864		1					
2	608	3918	31059	190410	2267073		2	67.19	87.26	76.41	97.65	1480.08
4	424	1755	11470	79170	715310		4	71.7	111.62	135.39	120.25	158.47
8	374	1109	4220	30643	210488		8	56.68	79.13	135.9	129.18	169.92
16	323	730	2785	11514	83500		16	57.89	75.96	75.76	133.07	126.04
32	294	649	1792	6372	36665		32	54.93	56.24	77.71	90.35	113.87
64	367	576	1454	4732	18416		64	40.05	56.34	61.62	67.33	99.55
128	454	571	1341	3749	11462		128	40.42	50.44	54.21	63.11	80.34
256	516		1423	4571	12469		256	43.99		47.12	41.01	45.96

d = 3									
Time, ms					Efficiency, %				
n	32	64	128	256	n	32	64	128	256
procs					procs				
1	345	6916	212756	8140475	1	–	–	–	–
2	263	3446	111121	2955568	2	31.18	100.7	91.46	175.43
4	175	1731	45085	1055536	4	32.38	99.85	123.97	223.74
8	129	888	12420	358084	8	23.92	96.98	230.43	310.48
16	86	490	4726	127972	16	20.08	87.43	293.45	417.41
32	68	290	2466	66859	32	13.14	73.7	275.08	389.54
64	58	316	1495	17918	64	7.85	33.15	224.3	719.55
128	433	184	945	7554	128	−0.16	28.81	176.49	847.75

5 Parallel algorithm of multiplication of polynomials of many variable integers in a ring by means of discrete transformation of Fourier

In work [1] the consecutive algorithm of multiplication of polynomials of one variable by means of DFT is described.

It is offered to calculate product of polynomials in r final fields $Z_{p_0}, \ldots, Z_{p_{r-1}}$ with the subsequent recoverong of result in Z with the Chinese remainder theorem.

If the number of variables in a polynomial more than one such algorithm can be carried out on several processors by parallelization calculations of direct and inverse DFTs and calculations over prime modules.

Let f, g - polynomials in a ring $Z[x_0, x_1, \ldots, x_{d-1}]$.

$$f = \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \ldots \sum_{i_d=0}^{n_d-1} f_{i_1 i_2 \ldots i_n} x_1^{i_1} x_2^{i_2} \ldots x_d^{i_d},$$

$$g = \sum_{i_1=0}^{m_1-1} \sum_{i_2=0}^{m_2-1} \dots \sum_{m_d=0}^{n_d-1} g_{i_1 i_2 \dots i_n} x_1^{i_1} x_2^{i_2} \dots x_d^{i_d}.$$

Let a_f, a_g - maximum on the coefficients of polynomials f and g accordingly. We designate $n = n_0 \cdot \ldots \cdot n_{d-1}$, $m = m_0 \cdot \ldots \cdot m_{d-1}$. Then $h = \min\{n, m\} \cdot a_f a_g$ - the top estimation of maxixmum on the coefficient in product of polynomials f and g. Then necessary for restoration of result the number of 32-bit prime modules $r = \lceil \log_{2^{32}} h \rceil$.

Let $s_i = 2^{S_i}$ - the minimum natural number, greater than $n_i + m_i$. Let there is a computer with $k = 2^K$ processors with numbers of $0, \dots, k-1$. We result parallel algorithm of multiplication of polynomials.

1. Processor with number 0 defines the necessary number of prime modules $p_0, \dots, p_{r-1}$. Let $r = 2^R$. We get $t = r/k$. In case $r < k$ sending of polynomials is executed only to r the first processors.

2. Processor 0 sends polynomials f, g and prime numbers $p_0, \dots, p_{r-1}$ to the processor with number $i = 0, \dots, k-1$. Following steps are carried out in parallel on each processor.

3. On the processor with number i product of polynomials $h_{i/t} = fg$ on modules $p_{i/t}, \dots, p_{(i+1)/t-1}$ by means of DFT is calculated. If $r < k$ each processor uses $k/r - 1$ of free processors for parallelization of DFT calculations.

4. Splitting arrays of coefficients for polynomial $h_{i/t} = fg$ on k parts $h_{i/t}^0, \dots, h_{i/t}^{k-1}$ is carried out.

5. Send parts $h_{i/t}^q$ to the processor with number q.

6. Receive $h_{i/t}^q$ on the processor with number q.

7. Recovering h_q polynomial coefficients h is carried out.

8. On the processor 0 polynomial assemblage h is carried out.

9. End of calculations.

6 Conclusion

In the given work comparison of two approaches has been spent to the organization of polynomial arithmetics over a ring of integers.

In section 2 the problem of multiplication of two polynomials with degree m, with the coefficients length w of machine words was considered. Theoretical and experimental comparison of following algorithms was spent:

0. Standard algorithm of multiplication of numbers and polynomials (PSS) with complexity $O(m^2 w^2)$.

1. Karatsuba's algorithm for multiplication of numbers and standard algorithm of multiplication of polynomials (PSK) with complexity $O(m^2 w^{\log_2 3})$.

2. Standard algorithm of multiplication of numbers and Karatsuba's algorithm for multiplication of polynomials (PKS) with complexity $O(m^{\log_2 3} w^2)$.

3. Karatsuba's algorithm for multiplication both numbers and polynomials (PKK) with complexity $O(m^{\log_2 3} w^{\log_2 3})$

4. The algorithm of multiplication using DFT (PF) with complexity $O(w(m \log_2 m) + mw^2)$. The best algorithm at increase in degree of polynomials was an algorithm 4. On the average the relative difference in theoretical and experimental estimations makes 35.86 %.

In section 3 algorithms of multiplication of polynomial matrices of the size n which elements are degree polynomials m with the coefficients length w of machine words were considered:

0. Modular algorithm of multiplication of the matrices, using CRT both for polynomials, and for their factors (MCC) with complexity $O(n^3 m + n^2(wm^2 + mw^2))$.

1. Modular algorithm of multiplication of the matrices, Fourier using fast transformation for polynomials and CRT for their factors (MFC) with complexity $O(n^3 m + n^2(wm \log_2 m + mw^2))$.

The best algorithm at increase of degree of polynomials in a matrix is MFC. At the fixed degree of polynomials and increasing n the best algorithm is MCC by results of theoretical and experimental comparison. Average relative distinction in theoretical and experimental comparison makes 24.37 %.

At a theoretical estimation of algorithms of calculation of a determinant, a characteristic polynomial and the adjoint matrix, for the matrices which elements are polynomials of one variable with the integer coefficients, following expressions of complexity have been received:

Table 12

Theoretical expression of complexity for algorithms of calculation of a determinant, a characteristic polynomial and the adjoint matrix for the matrices which elements are polynomials of one variable with the integer coefficients. n - the size of a matrix, s - the maximum degree of a polynomial in a matrix, α - maximum on the module of polynomial's coefficient in a matrix, r - the number of the prime numerical modules necessary for calculation of a determinant and the adjoint matrix, r_1 - the number of the prime numerical modules necessary for calculation of a characteristic polynomial, $h = 2^{32}$

Algorithm	CRT+CRT	CRT+DFT
Determinant	$n^2 r(7s\lceil \log_h \alpha \rceil + ns^2) +$ $+8n^4 rs + 2n^2 s^2 \cdot r + 2r^2 sn$	$n^2 r(7s\lceil \log_h \alpha \rceil + 9ns \log_2(ns)) +$ $8n^4 rs + 9ns \log_2(ns) \cdot r + 2r^2 sn$
Char. polynomial	$n^2 r_1(7s\lceil \log_h \alpha \rceil + ns^2) +$ $8n^4 r_1 s + n(2n^2 s^2 \cdot r_1 + 2r_1^2 sn)$	$n^2 r_1(7s\lceil \log_h \alpha \rceil + 9ns \log_2(ns)) +$ $+8n^4 r_1 s + n(9ns \log_2(ns) \cdot r + 2r_1^2 sn)$
Adjoint matrix	$n^2 r(7s\lceil \log_h \alpha \rceil + ns^2)$ $+8n^4 rs + n^2(2n^2 s^2 \cdot r + 2r^2 sn)$	$n^2 r(7s\lceil \log_h \alpha \rceil + 9ns \log_2(ns))$ $+8n^4 rs + n^2(9ns \log_2(ns) \cdot r + 2r^2 sn)$

From Table 12 it is clear that the best results the DFT-arithmetic has been shown at a calculation of the adjoint matrix. In problems of calculation of a determinant and a characteristic polynomial the algorithms using DFT, lose the efficiency with growth of the sizes of a matrix.

The algorithm of parallel calculation DFT in a finite field has shown high scalability on the number of processors from 2 to 256. Algorithm acceleration varies from -0.16 % up to 1480 %. It is possible to explain such acceleration to that at increase in the number of processors the computing problem on everyone becomes small enough for effective work with the data in cache-memory of the processor.

The further direction of researches is the implementation of parallel DFT-algorithms for multiplication of matrices, calculations of a determinant, a characteristic polynomial and the adjoint matrix.

References

1. *Noden P., Kitte K.* Algebraic algorithmic (with exercises and solutions)/ the traslation from French, The World, 1999.

2. *Moreno Maza M., Xie Y.* FFT-based Dense polynomial Arithmetic on Multi-cores // HPCS 2009: post-conference proceedings. 2009.

3. *Moreno Maza M., Xie Y.* Balanced Dense polynomial Multiplication on Multi-cores // Proceedings of Parallel and Distributed Computing, Applications and Technologies (PDCAT). 2009.

4. *Knut D.E.* The art of programming. V. 2. Seminumerical algorithms. The publishing house «Williams», 2001.

5. *Kormen, Thomas S., Lejzerson, Rivest Ch., Shtojn R., Klifford* Algorithms: construction and the analysis, 2 edition/ the translation from English, The Publishing house «Williams», 2005.

6. *Pereslavtseva O.N.* The method of calculation of a characteristic polynomial of a matrix// Tambov University Reports. Series Natural and Technical Sciences. 2008. V. 13. Issue 1. P. 131-133.

7. *Lapaev A.O.* Comparison of algorithms of multiplication of polynomials // International conference polynomial Computer Algebra. St. Petersburg. PDMI RAS. 2008. P. 39 - 40.

8. *Lapaev A.O.* Parallel computation of discrete Fourier transform of a polynomial in a finite field//Materials of 9th international conference-seminar High-efficiency parallel calculations on cluster systems. Vladimir, 2009. P. 272-273.

9. *Lapaev A.O.* About calculation of multidimensional discrete transformation of Fourier in a finite field // Tambov University Reports. Natural and Technical Sciences. 2009. V. 14. Issue 4. P. 729-731.

10. *Lapaev A.O.* Algorithms for polynomial matrices with use DFT // International conference polynomial Computer Algebra. St. Petersburg. PDMI RAS. 2009. P. 141-146.

11. *Lapaev A.O.* On calculation of determinant and a characteristic polynomial of polynomial matrix with use of discrete Fourier transform // Tambov University Reports. Natural and Technical Sciences. 2009. V. 14. Issue 1. P. 281-282.

12. *Malaschonok G I., Valeev Yu. D., Lapaev A. O.* On the choice of multiplication algorithm for polynomials and polynomial matrices // Zapiski POMI. 2009. V. 373. P. 157-188.

GRATITUDES: Supported by the Sci. Program Devel. Sci. Potent. High. School, RNP 2.1.1.1853.

Accepted for publication 7.06.2010.

ДПФ ДЛЯ ПОЛИНОМОВ В ПАРАЛЛЕЛЬНЫХ АЛГОРИТМАХ

© **Алексей Олегович Лапаев**

Тамбовский государственный университет им. Г.Р. Державина, Интернациональная, 33, Тамбов, 392000, Россия, аспирант кафедры компьютерного и математического моделирования, e-mail alapaev@gmail.com

Ключевые слова: полиномы; дискретное преобразование Фурье; параллельный алгоритм; метод гомоморфных образов; кластер.

Рассматриваются последовательные и параллельные алгоритмы для полиномиальной арифметики, основанные на дискретном преобразовании Фурье (ДПФ). Обсуждаются алгоритмы для умножения полиномов. Приведены последовательные алгоритмы для полиномиальных матриц. Каждый алгоритм, основанный на ДПФ, сравнивается с аналогичным алгоритмом, использующим китайскую теорему об остатках. В последней части работы приведены параллельные алгоритмы для вычисления ДПФ и умножения полиномов многих переменных. Приведены результаты экспериментов на кластере МВС100К в МСЦ РАН.

UDC 519.688

FAST MATRIX DECOMPOSITION IN PARALLEL COMPUTER ALGEBRA

© **Gennadi Ivanovich Malaschonok**
Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Doctor of Physics and Mathematics, Professor of Mathematical Analysis Department, e-mail: malaschonok@ya.ru

Key words: fast algorithms; matrix decomposition; parallel algorithms; computer algebra.

The new algorithms for finding matrix decomposition and matrix inversion in arbitrary fields are described. For the commutative domains the algorithm for finding adjoint matrices is proposed. These algorithms have the same complexity as matrix multiplication and do not require pivoting. For singular matrices they allow to obtain a nonsingular block of the biggest size. The proposed algorithms are pivot-free, and do not change the matrix block structure. They are suitable for parallel hardware implementation.

1 Introduction

One of the popular linear algebra method is LU matrix decomposition. A lot of different implementations are well known for this decomposition. But the LU decomposition requires pivoting. With partial pivoting it has the form $PA = LU$ and with full pivoting (Trefethen and Bau) it has the form $PAQ = LU$, where L and U are the lower- and the upper- triangular matrices, P and Q are the permutation matrices.

Another well known decomposition is the decomposition of inverse matrix, which is based on the Schur complement trick. Let $\mathcal{A} = \begin{pmatrix} A & C \\ B & D \end{pmatrix}$ be an invertible matrix with invertible block A, then the inverse matrix $\mathcal{A}^{-1}$ can be written in the form

$$\begin{pmatrix} I & -A^{-1}C \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & (D - BA^{-1}C)^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -B & I \end{pmatrix} \begin{pmatrix} A^{-1} & 0 \\ 0 & I \end{pmatrix}.$$

In these algorithms it is assumed that principal minors are invertible and the leading elements are nonzero as in the most of the direct algorithms for matrix inversion. Fast matrix multiplication and fast block matrix inversion were discovered by Strassen [1]. If we used fast matrix multiplication then we get the recursive algorithm for matrix inversion which has the same complexity as the algorithm of matrix multiplication.

In a general case it is necessary to find suitable nonzero elements and to perform permutations of matrix columns or rows. Bunch and Hopkroft suggested such algorithm with full pivoting for matrix inversion [2].

There are known other recursive methods for adjoint and inverse matrix computation, which have the complexity of matrix multiplications ([3]-[5]).

The permutation operation is not a very difficult operation in the case of sequential computations by one processor, but it is a difficult operation in the case of parallel computations, when different blocks of a matrix are disposed in different processors. A matrix decomposition without permutations is needed for parallel computation for construction of efficient and fast computational schemes.

The problem of obtaining pivot-free algorithm was studied in [6], [7] by S.Watt. He presented the algorithm that is based on the following identity for a nonsingular matrix: $A^{-1} = (A^T A)^{-1} A^T$. Here A^T is the transposed matrix to A and all principal minors of the matrix $A^T A$ are nonzero. This method is useful for making an efficient parallel program with the help of Strassen's fast decomposition of inverse matrix for dense nonsingular matrix over the field of zero characteristic when field elements are represented by the float numbers. Other parallel matrix algorithms are developed in [8] - [11].

Another form of matrix A decomposition is Bruhat decomposition $A = VwU$, where V and U are nonsingular upper triangular matrices and w is a matrix of permutation. French mathematician Francois Georges René Bruhat was the first who worked with matrix decomposition in this form. Bruhat decomposition plays an important role in algebraic group. The generalized Bruhat decomposition was introduced and developed by D.Grigoriev [1], [12]. He uses the Bruhat decomposition in the form $A = VwU$, where V and U are upper triangular matrices but they may be singular when the matrix A is singular. Sparsity pattern of triangular factors of the Bruhat decomposition of a nonsingular matrix over a field was analyzed in [13] and [14].

This paper is devoted to the construction of matrix decomposition methods in a common case of singular matrices in a field of arbitrary characteristic and in the domain.

For the matrix over the field two decompositions will be constructed which have the forms $LAU = E$ and $FA = H$. For the matrix over the domain one decompositions will be constructed of the form: $GA = dH$. Where L and U are lower and upper nonsingular triangular matrices, F and G is a nonsingular matrix, d is a determinant of some nonsingular block of matrix A which size is equal $rank(A)$ and equalities $rank(E) = rank(H) = rank(A)$ hold.

In the case of full rank matrix A the matrix E and H are permutation matrices, $UE^T L$ and $H^T F$ are inverse matrices for matrix A, and $H^T G$ is the adjoint matrix for the matrix A.

These algorithms have the same complexity as matrix multiplication and do not require pivoting. For singular matrices they allow to obtain a nonsingular block of the biggest size and the echelon form and the kernel of matrix.

The preliminary variants of these algorithms were developed in [15], [16] and [17].

The rest of the paper is organized as follows. Section 2 provides some necessary background and notations. Section 3 presents the algorithm of LEU decomposition. Section 4 presents the fast matrix decomposition in the field and computation of the inverse matrix. Section 5 presents the fast matrix decomposition in the commutative domain and computation of ajoit matrix. In the Appendix 6 we dispose the proof of the theorem 1.

2 Preliminaries

We introduce some notations that will be used in the following sections.

Let F be a field, $F^{n\times n}$ be an $n\times n$ matrix ring over F, S_n be a permutation group of n elements. Let P_n be a multiplicative semigroup in $F^{n\times n}$ consisting of matrices A having exactly rank(A) nonzero entries, all of them equal to 1. We call P_n the permutation semigroup because it contains the permutation group of n elements S_n and all their truncated matrix.

The semigroup $D_n \subset P_n$ is formed by the diagonal matrices. So $|D_n| = 2^n$ and the identity matrix $\mathbf{I}$ is the identity element in D_n, S_n and P_n.

Let $W_{i,j} \in P_n$ be a matrix, which has only one nonzero element in the position (i,j). For an arbitrary matrix E of P_n, which has the rank $n-s$ $(s=0,..n)$ we shall denote by $i_{\overline{E}} = \{i_1,..,i_s\}$ the ordered set of zero row numbers and $j_{\overline{E}} = \{j_1,..,j_s\}$ the ordered set of zero column numbers.

Definition 1 *Let $E \in P_n$ be the matrix of the rank $n-s$, let $i_{\overline{E}} = \{i_1,..,i_s\}$ and $j_{\overline{E}} = \{j_1,..,j_s\}$ are the ordered set of zero row numbers and zero columns number of the matrix E. Let us denote by $\overline{E}$ the matrix*

$$\overline{E} = \sum_{k=1,..s} W_{i_k,j_k}$$

and call it the complimentary matrix for E. For the case $s=0$ we put $\overline{E}=0$.

It is easy to see that $\forall E \in P_n:\ E+\overline{E} \in S_n$, and $\forall I \in D_n:\ I+\overline{I} = \mathbf{I}$. Therefore the map $I \mapsto \overline{I} = \mathbf{I} - I$ is the involution and we have $I\overline{I} = 0$. We can define the a partial order on D_n: $I < J \Leftrightarrow J - I \in D_n$. For each matrix $E \in P_n$ we shall denote by

$$I_E = EE^T \text{ and } J_E = E^TE$$

the diagonal matrix: $I_E, J_E \in D_n$. The unit elements of the matrix I_E show nonzero rows of the matrix E and the unit elements of the matrix J_E show nonzero columns of the matrix E. Therefore we have several zero identities:

$$E^T\overline{I}_E = \overline{I}_E E = E\overline{J}_E = \overline{J}_E E^T = 0. \tag{1}$$

For any pair $I, J \in D_n$ let us denote the subset of matrices $F^{n\times n}$

$$F^{n\times n}_{I,J} = \{B : B \in F^{n\times n}, IBJ = B\}.$$

We call them (I,J)-zero matrix. It is evident that $F^{n\times n} = F^{n\times n}_{\mathbf{I},\mathbf{I}}$, $0 \in \cup_{I,J} F^{n\times n}_{I,J}$ and if $I_2 < I_1$ and $J_2 < J_1$ then $F^{n\times n}_{I_2,J_2} \subset F^{n\times n}_{I_1,J_1}$.

Definition 2 *We shall call the factorization of the matrix $A \in F^{n\times n}_{I,J}$*

$$A = L^{-1}EU^{-1}, \tag{2}$$

LEU-decomposition if $E \in P_n$, L is a nonsingular lower triangular matrix, U is an upper unitriangular matrices and

$$L - \overline{I}_E \in F^{n\times n}_{I,I_E}, \quad U - \overline{J}_E \in F^{n\times n}_{J_E,J}. \tag{3}$$

If (2) is the LEU-decomposition we shall write

$$(L,E,U) = \mathcal{LU}(A),$$

Sentence 1 *Let* $(L, E, U) = \mathcal{LU}(A)$ *be the LEU -decomposition of matrix* $A \in F_{I,J}^{n\times n}$ *then*

$$L = \overline{I}_E + ILI_E,\ U = \overline{J}_E + J_E U J,\ E \in F_{I,J}^{n\times n}, \tag{4}$$

$$L^{-1} = \overline{I}_E + L^{-1}I_E,\ U^{-1} = \overline{J}_E + J_E U^{-1}.$$

Proof 1 *The first and second equalities follows from (3). To prove the property of matrix* E *we use the commutativity of diagonal semigroup* D_n*:*

$$E = LAU = (\overline{I}_E + ILI_E)IAJ(\overline{J}_E + J_E U J) = I(\overline{I}_E + LI_E I)A(\overline{J}_E + JJ_E U)J.$$

To prove the property of matrix L^{-1} *let us consider the identity*

$$\mathbf{I} = L^{-1}L = L^{-1}(\overline{I}_E + LI_E) = L^{-1}\overline{I}_E + \mathbf{I}I_E.$$

Therefore $L^{-1}\overline{I}_E = \overline{I}_E$ *and* $L^{-1} = L^{-1}(\overline{I}_E + I_E) = \overline{I}_E + L^{-1}I_E$*. The proof of the matrix* U^{-1} *property may be obtained similarly.*

Sentence 1 states the property of matrix E, which may be written in the form $I_E < I$, $J_E < J$. We shall call it *the property of immersion.*

Examples.

For any matrix $I \in D_n$, $E \in P_n$, $0 \neq a \in F$ the product $(aI + \overline{I})I$ $\mathbf{I}$ is a LEU decompositions of matrix aI and the product $(aI_E + \overline{I}_E)E$ $\mathbf{I}$ is a LEU decompositions of the matrix aE.

3 Algorithm of *LEU* decomposition

Theorem 1 *For any matrix* $A \in F^{n\times n}$ *of size* $n = 2^k$, $k \geqslant 0$ *a LEU -decomposition exists. For computing such decomposition it is enough to compute 4 LEU -decompositions and 17 multiplications for the matrices of size* $n = 2^{k-1}$.

The proof of this theorem is in the Appendix.

Theorem 2 *For any matrix* A *of size* $s, (s \geqslant 1)$, *an algorithm of LEU -decomposition which has the same complexity as matrix multiplication exists.*

Proof 2 *We have proved an existence of LEU -decomposition for matrices of size* $2^k, k > 0$. *Let* $A \in F_{I,J}^{s\times s}$ *be a matrix of size* $2^{k-1} < s < 2^k$, A' *be a matrix of size* 2^k, *which has in the left upper corner the submatrix equal* A *and all other elements equal zero. We can construct LEU -decomposition of matrix* A'*:* $(L', E', U') = \mathcal{LU}(A')$. *According to the Sentence 1 the product* $L'A'U' = E'$ *has the form*

$$\begin{pmatrix} L & 0 \\ 0 & \mathbf{I} \end{pmatrix}\begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} U & 0 \\ 0 & \mathbf{I} \end{pmatrix} = \begin{pmatrix} E & 0 \\ 0 & 0 \end{pmatrix}.$$

Therefore $LAU = E$ *is a LEU decomposition of matrix* A.

The total amount of matrix multiplications in (7)-(15) is equal to 17 and total amount of recursive calls is equal to 4. We do not consider multiplications of the permutation matrices.

We can compute the decomposition of the second order matrix by means of 5 multiplicative operations. Therefore we get the following recurrent equality for complexity

$$t(n) = 4t(n/2) + 17M(n/2), t(2) = 5.$$

Let γ *and* β *be constants,* $3 \geqslant \beta > 2$, *and let* $M(n) = \gamma n^{\beta} + o(n^{\beta})$ *be the number of multiplication operations in one* $n \times n$ *matrix multiplication.*

After summation from $n = 2^k$ *to* 2^1 *we obtain*

$$17\gamma(4^0 2^{\beta(k-1)} + \ldots + 4^{k-2}2^{\beta 1}) + 4^{k-2}5 = 17\gamma\frac{n^{\beta} - 2^{\beta-2}n^2}{2^{\beta} - 4} + \frac{5}{16}n^2.$$

Therefore the complexity of the decomposition is

$$\sim \frac{17\gamma n^{\beta}}{2^{\beta} - 4}.$$

If A is an invertible matrix, then $A^{-1} = UE^TL$ and a recursive block algorithm of matrix inversion is written in the expressions (7)-(15). This algorithm has the complexity of matrix multiplications.

4 Algorithm with one-sided decomposition

Let us introduce the set of h-matrices in the ring $F^{n\times n}$. We say that a matrix H is of h-type if for some matrix $E \in P_n$ the two equations $H = I_E H$ and $E = HJ_E$ take place. We call E *the main part* of the h-type matrix H. In other words, the sets of zero rows of matrices H and E coincide and each nonzero column of matrix E stands at the same place in the matrix H. In particular, if H is nonsingular, then $H \in S_n$. The nonzero columns of matrix E are called *the main columns* of matrix H.

Let $A \in F^{n\times n}$ be a matrix of $\operatorname{rank}(A) = r \leqslant n$. We wish to obtain a nonsingular matrix R with the following properties:

(1) $RA = H$ and H is a matrix of h-type with main part $E \in P_n$.

(2) If $A = \overline{I}A$, $I \in D_n$, $\operatorname{rank}\overline{I} = r$, then $R = I + \overline{I}R\overline{I}$.

If A is invertible, then $H^TH = \mathbf{I}$ and $A^{-1} = H^TR$. As this algorithm is a generalization of the inversion algorithm, we call it the *H-inversion algorithm.*

One recursive step $RA = H$ may be written as three matrix multiplications

$$R_1A = A^1,\ R_2A^1 = A^2,\ R_3A^2 = A^3,\ R = R_3R_2R_1,\ A^3 = H.$$

It is easy to produce the required matrices R when the size of matrix A equals 2. Let the size of the matrix A equals $2n$ and matrices A and A^i have the following block form:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad A^i = \begin{pmatrix} A^i_{11} & A^i_{12} \\ A^i_{21} & A^i_{22} \end{pmatrix}, \ (i = 1, 2, 3).$$

4.1 The first substep

Let $R_{11}A_{11} = H_{11}$ and H_{11} be the h-type matrix with the main part E_{11}. We denote $J_{11} = E_{11}^T E_{11}$, $I_{11} = E_{11}E_{11}^T$ and put

$$R_1 = \begin{pmatrix} \mathbf{I} & 0 \\ -A_{21}E_{11}^T & \mathbf{I} \end{pmatrix} \begin{pmatrix} R_{11} & 0 \\ 0 & \mathbf{I} \end{pmatrix} A \quad \text{and} \quad R_1 A = A^1.$$

Then we obtain the following blocks of the matrix A^1:

$$A_{11}^1 = H_{11},\ A_{12}^1 = R_{11}A_{12},\ A_{21}^1 = A_{21}(\mathbf{I} - E_{11}^T H_{11}),\ A_{22}^1 = A_{22} - A_{21}E_{11}^T A_{12}^1.$$

Let us note that $A_{21}^1 = A_{21}^1 \overline{J}_{11}$, because in the place of each nonzero column of matrix E_{11} in the matrix $\mathbf{I} - E_{11}^T H_{11}$ the zero column stands.

4.2 The second substep

Let the matrices R_{12} and R_{21} satisfy the equations

$$R_{12}A_{12}^1 = H_{12}, \quad R_{21}A_{21}^1 = H_{21},$$

where H_{12} and H_{21} are the h-type matrices with the main parts E_{12} and E_{21} respectively. Let us denote $B_{22}^1 = R_{21}A_{22}^1$. Consider the diagonal matrices $J_{12} = E_{12}^T E_{12}$, $J_{21} = E_{21}^T E_{21}$, $I_{12} = E_{12}E_{12}^T$, $I_{21} = E_{21}E_{21}^T$ and put

$$R_2 = \begin{pmatrix} \mathbf{I} & -A_{11}^1 E_{21}^T \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ -B_{22}^1 E_{12}^T & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} - I_{11}A_{12}^1 E_{12}^T & 0 \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} R_{12} & 0 \\ 0 & R_{21} \end{pmatrix} =$$

$$= \begin{pmatrix} (\mathbf{I} - I_{11}A_{12}^1 E_{12}^T + A_{11}^1 E_{21}^T B_{22}^1 E_{12}^T)R_{12} & -A_{11}^1 E_{21}^T R_{21} \\ -B_{22}^1 E_{12}^T R_{12} & R_{21} \end{pmatrix}.$$

Using the identities $H_{12} = \overline{I}_{11}H_{12}$, $I_{11}E_{12} = 0$, $E_{12}^T I_{11} = 0$, and the fact, that $R_{12} = I_{11} + \overline{I}_{11}R_{12}\overline{I}_{11}$ and $H_{11} = I_{11}H_{11}$

$$R_{12}H_{11} = H_{11},\ E_{12}^T R_{12}H_{11} = 0, E_{12}^T R_{12}A_{12}^1 = E_{12}^T H_{12},$$

we get the block

$$A_{12}^2 = R_{12}A_{12}^1 + (-I_{11}A_{12}^1 + A_{11}^1 E_{21}^T B_{22}^1)E_{12}^T H_{12} - A_{11}^1 E_{21}^T B_{22}^1 =$$
$$H_{12} + I_{11}A_{12}^1 - I_{11}A_{12}^1 E_{12}^T H_{12} + A_{11}^1 E_{21}^T B_{22}^1 (E_{12}^T H_{12} - \mathbf{I}) =$$
$$H_{12} + (I_{11}A_{12}^1 - A_{11}^1 E_{21}^T B_{22}^1)(\mathbf{I} - E_{12}^T H_{12}),$$

and other blocks of matrix A^2:

$A_{11}^2 = H_{11}(\mathbf{I} - E_{21}^T H_{21})$,
$A_{12}^2 = H_{12} + (I_{11}A_{12}^1 - A_{11}^1 E_{21}^T B_{22}^1)(\mathbf{I} - E_{12}^T H_{12})$,
$A_{21}^2 = H_{21}$,
$A_{22}^2 = B_{22}^1(\mathbf{I} - E_{12}^T R_{12}A_{12}^1) = B_{22}^1(\mathbf{I} - E_{12}^T H_{12})$.

Let us note that these blocks have the properties $A_{11}^2 = A_{11}^2 \overline{J}_{21}$, $A_{21}^2 = A_{21}^2 \overline{J}_1$, $A_{12}^2 - E_{12} = (A_{12}^2 - E_{12})\overline{J}_{12}$, $A_{22}^2 = A_{22}^2 \overline{J}_{12}$. We use them in the following section.

4.3 The third substep

Let the matrices R_{22} satisfy the equation

$$R_{22}A^2_{22} = H_{22},$$

where H_{22} is the h-type matrix with the main part E_{22}.

Let us consider the diagonal matrices $J_{22} = E^T_{22}E_{22}$, $I_{22} = E_{22}E^T_{22}$ and put

$$R_3 = \begin{pmatrix} \mathbf{I} & -A^2_{12}E^T_{22} \\ 0 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{I} - I_{21}A^2_{22}E^T_{22} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 \\ 0 & R_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & -A^2_{12}E^T_{22}R_{22} \\ 0 & (\mathbf{I} - I_{21}A^2_{22}E^T_{22})R_{22} \end{pmatrix}.$$

Then we obtain the matrix $A_3 = R_3A_2$ with the blocks:

$$A^3_{11} = A^2_{11},\ A^3_{21} = A^2_{21},\ A^3_{12} = A^2_{12}(\mathbf{I} - E^T_{22}H_{22}),\ A^3_{22} = H_{22} + I_{21}A^2_{22}(\mathbf{I} - E^T_{22}H_{22})$$

that have the properties $A^3_{12} = A^3_{12}\overline{J}_{22}$, $A^3_{22} - E_{22} = (A^3_{22} - E_{22})\overline{J}_{22}$.

4.4 The result

We obtain the matrix

$$R = R_3R_2R_1 = \begin{pmatrix} L + FG & -FR_{21} \\ -MG & MR_{21} \end{pmatrix}.$$

Here we use the notation

$L = (\mathbf{I} - I_{11}A^1_{12}E^T_{12})R_{12}R_{11}$,
$M = (\mathbf{I} - I_{21}A^2_{22}E^T_{22})R_{22}$,
$F = (A^1_{11}E^T_{21} + A^2_{12}E^T_{22}R_{22})$,
$G = R_{21}(A^1_{22}E^T_{12}R_{12} + A_{21}E^T_{11})R_{11}$.
$A^1_{12} = R_{11}A_{12}$,
$A^1_{21} = A_{21}(\mathbf{I} - E^T_{11}H_{11})$,
$A^1_{22} = A_{22} - A_{21}E^T_{11}A^1_{12}$,
$A^2_{22} = R_{21}A^1_{22}(\mathbf{I} - E^T_{12}H_{12})$,
$A^2_{12} = H_{12} + I_{11}A^1_{12}(\mathbf{I} - E^T_{12}H_{12}) - H_{11}E^T_{21}A^2_{22}$.

And we obtain the equation $RA = H$ with h-type matrix H which has the main part $E = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}$.

4.5 The important particular cases

We can outline two important particular cases.

In the first case the matrix A has an invertible block A_{11}. In this case we obtain $H_{11} = H_{22} = R_{12} = R_{21} = \mathbf{I}$, $H_{12} = H_{21} = E_{12} = E_{21} = 0$, $A^1_{22} = A_{22} - A_{21}A^1_{12}$, $A^2_{22} = R_{21}A^1_{22}$, $A^2_{12} = A^1_{12} = R_{11}A_{12}$, $L = R_{11}$, $M = R_{22}$, $F = A^2_{12}R_{22}$, $G = A_{21}R_{11}$,

$$R = \begin{pmatrix} R_{11} + R_{11}A_{12}R_{22}A_{21}R_{11} & -R_{11}A_{12}R_{22} \\ -R_{22}A_{21}R_{11} & R_{22} \end{pmatrix}.$$

More over, if matrix A is invertible, then $R = A^{-1}$.

In the second case the matrix A has zero block $A_{11} = 0$ and two invertible blocks A_{21} and A_{12}. In this case we obtain $H_{11} = H_{22} = 0, R_{11} = R_{22} = H_{12} = H_{21} = \mathbf{I}$, $A^1_{12} = A_{12}$, $A^1_{21} = A_{21}$, $A^1_{22} = A_{22}$, $A^2_{22} = 0$, $A^2_{12} = H_{12}$, $L = R_{12}R_{11}$, $M = \mathbf{I}$, $F = 0$, $G = R_{21}A_{22}R_{12}$,

$$R = \begin{pmatrix} R_{12} & 0 \\ -R_{21}A_{22}R_{12} & R_{21} \end{pmatrix}.$$

5 Matrix decomposition in the commutative domain R

The mapping $A_{ext}: R^{n\times n} \times (R\backslash 0) \to (R^{n\times n})^3 \times (R\backslash 0)$

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

with $n = 2^k$ we call the extended adjoint mapping of the couple (M, d_0) if it recursive defined as follows.

For $M = 0$: $A_{ext}(M, d_0) = (d_0\mathbf{I}, 0, 0, d_0)$.

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

For $k > 0$ and $M \neq 0$ we have to divide matrix M into four equal blocks $M = (M_{ij})$, $i, j \in \{1, 2\}$. Let

$$(A_{11}, S_{11}, E_{11}, d_{11}) = A_{ext}(M_{11}, d_0),$$

we denote

$$M^1_{12} = A_{11}M_{12}/d_0,\ M^1_{21} = M_{21}Y_{11}/d_0,\ M^1_{22} = (M_{22} - M_{21}E^T_{11}M^1_{12})/d_0.$$

Let

$$(A_{12}, S_{12}, E_{12}, d_{12}) = A_{ext}(\bar{I}_{11}M^1_{12}, d_{11}),$$

$$(A_{21}, S_{21}, E_{21}, d_{21}) = A_{ext}(M^1_{21}, d_{11}).$$

Denote

$$M^2_{22} = A_{21}M^1_{22}Y_{12}/(d_{11})^2, d_s = d_{21}d_{12}/d_{11}.$$

Let

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

Denote

$$M^2_{11} = S_{11}Y_{21}/d_{11},\ M^2_{12} = S_{12}d_{21} + (I_{11}M^1_{12}d_{11}d_{21} - S_{11}E^T_{21}A_{21}M^1_{22})Y_{12}/(d_{11})^2.$$

$$M^3_{12} = M^2_{12}Y_{12}/(d_sd_{11}),\ M^3_{22} = S_{22} + I_{21}M^2_{22}Y_{12}/d_s,$$

$$L = (\mathbf{I}d_{11} - I_{11}M^1_{12}E^T_{12})A_{12}A_{11}d_{22}/(d_{11})^2,$$

$$Q = (\mathbf{I}d_{12}d_{21} - I_{21}M^2_{22}E^T_{22}d_{11})A_{22}/d_s,$$

$$F = S_{11}E^T_{21}d_{22} + M^2_{12}E^T_{22}A_{22}/d_s,$$

$$G = A_{21}(M^1_{22}E^T_{12}A_{12}d_0 + M_{21}E^T_{11}d_{12}d_{11})A_{11}/(d^2_{11}d_0),$$

$$A = \begin{pmatrix} (L + FG/(d_{11}d_{21})/d_{12} & -FA_{21}/(d_{11}d_{21}) \\ -QG/(d_{11}d_{21}d_{12}) & QA_{21}/(d_{11}d_{21}) \end{pmatrix},$$

$$S = \begin{pmatrix} M_{11}^2 d_{22}/d_{21} & M_{12}^3 \\ S_{21} d_{22}/d_{21} & M_{22}^3 \end{pmatrix}, E = \begin{pmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{pmatrix}.$$

Then

$$(A, S, E, d_{22}) = A_{ext}(M, d_0).$$

We use here the notations

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

Theorem 3 *For arbitrary matrix* $M \in R^{n\times n}$ *the extended adjoint mapping* $(A, S, E, d) = A_{ext}(M, 1)$ *defines the extended adjoint nonsingular matrix* A, *the echelon matrix* S *and the matrix* E, *which have the property:* $AM = S$ *and* $dE = SJ_E$.

The proof is based on the algorithm with one-sided decomposition of the previous section. All division operations are based on the determinant identities [4] and give as a result the quotients which are the elements of the domain R or matrices over the domain R.

Let S be the echelon matrix obtained from the matrix M, $S_1 = E^T S$, then $E^T AM = E^T S$ and $S_1 J_E = dJ_E$. Let us write the matrix S_1 in the form $S_1 = S_0 + dJ$. It is easy to see that $S_0^2 = 0$. Therefore $(S_0 + dJ)(S_0 - d\bar{J}) = 0$, $rank(S_0 - d\bar{J}) = rank(\bar{J})$, $rank(S_1) = rank(J)$, so $rank(S_1) + rank(S_0 - d\bar{J}) = n$. That is why the columns of the matrix $S_0 + d\bar{J}$ generate the kernel of S_1, therefore they generate the kernel of M.

So we obtain the kernel of M:

$$kern(M) = span(E^T AM - d\mathbf{I}).$$

6 Conclusion

The algorithms for finding matrix decomposition and matrix inversion are described. These algorithms have the same complexity as matrix multiplication and do not require pivoting. For singular matrices they allow to obtain a nonsingular block of the biggest size. These algorithms may be used in any field, including real and complex numbers, finite fields and their extensions. The proposed algorithms are pivot-free, and do not change the matrix block structure. They are suitable for parallel hardware implementation.

7 Appendix

In this appendix we put the proof of the theorem 1.

Proof of Theorem 1

For the matrix of size 1×1, when $k = 0$, we can write the following LEU decompositions

$$\mathcal{LU}(0) = (1, 0, 1) \text{ and } \mathcal{LU}(a) = (a^{-1}, 1, 1),\ \text{if } a \neq 0.$$

Let us assume that for any matrix of size n we can write a LEU decomposition and let us given matrix $A \in F_{I,J}^{2n\times 2n}$ has the size $2n$. We shall construct a LEU decomposition of matrix A.

First of all we shall divide the matrices A, I, J and a desired matrix E into four equal blocks:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, I = \mathrm{diag}(I_1, I_2), J = \mathrm{diag}(J_1, J_2), E = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}, \tag{5}$$

and denote

$$I_{ij} = E_{ij}E_{ij}^T, \quad J_{ij} = E_{ij}^T E_{ij} \quad \forall i, j \in \{1, 2\}. \tag{6}$$

Let

$$(L_{11}, E_{11}, U_{11}) = \mathcal{LU}(A_{11}), \tag{7}$$

denote the matrices

$$Q = L_{11}A_{12}, \ B = A_{21}U_{11}, \tag{8}$$

$$A_{21}^1 = B\overline{J}_{11}, \ A_{12}^1 = \overline{I}_{11}Q, \ A_{22}^1 = A_{22} - BE_{11}^T Q. \tag{9}$$

Let

$$(L_{12}, E_{12}, U_{12}) = \mathcal{LU}(A_{12}^1) \text{ and } (L_{21}, E_{21}, U_{21}) = \mathcal{LU}(A_{21}^1), \tag{10}$$

denote the matrices

$$G = L_{21}A_{22}^1 U_{12}, \ A_{22}^2 = \overline{I}_{21}G\overline{J}_{12}. \tag{11}$$

Let us put

$$(L_{22}, E_{22}, U_{22}) = \mathcal{LU}(A_{22}^2), \tag{12}$$

and denote

$$W = (GE_{12}^T L_{12} + L_{21}BE_{11}^T), \ V = (U_{21}E_{21}^T G\overline{J}_{12} + E_{11}^T QU_{12}), \tag{13}$$

$$L = \begin{pmatrix} L_{12}L_{11} & 0 \\ -L_{22}WL_{11} & L_{22}L_{21} \end{pmatrix}, U = \begin{pmatrix} U_{11}U_{21} & -U_{11}VU_{22} \\ 0 & U_{12}U_{22} \end{pmatrix}. \tag{14}$$

We have to prove that

$$(L, E, U) = \mathcal{LU}(A). \tag{15}$$

As far as $L_{11}, L_{12}, L_{21}, L_{22}$ are low triangular nonsingular matrices and U_{11}, U_{12}, U_{21}, U_{22} are upper unitriangular matrices we can see in (10) that the matrix L is a low triangular nonsingular matrix and the matrix U is upper unitriangular.

Let us show that $E \in P_{2n}$. As far as $E_{11}, E_{12}, E_{21}, E_{22} \in P_n$ and $A_{11} = I_1 A_{11} J_1$, $A_{21}^1 = B\overline{J}_{11}$, $A_{12}^1 = \overline{I}_{11}Q$, $A_{22}^2 = \overline{I}_{21}G\overline{J}_{12}$ and due to the Sentence 1 we obtain $E_{11} = I_{11}E_{11}J_{11}$, $E_{21} = E_{21}\overline{J}_{11}$, $E_{12} = \overline{I}_{11}E_{12}$, $E_{22} = \overline{I}_{21}E_{22}\overline{J}_{12}$.

Therefore the unit elements in each of the four blocks of the matrix E are disposed in different rows and columns of the matrix E. So $E \in P_{2n}$, and next identities hold

$$E_{11}E_{21}^T = E_{11}J_{21} = J_{11}E_{21}^T = J_{11}J_{21} = 0, \tag{16}$$

$$E_{12}^T E_{11} = E_{12}^T I_{11} = I_{12}E_{11} = I_{12}I_{11} = 0, \tag{17}$$

$$E_{12}E_{22}^T = E_{12}J_{22} = J_{12}E_{22}^T = J_{12}J_{22} = 0, \tag{18}$$

$$E_{22}^T E_{21} = E_{22}^T I_{21} = I_{22}E_{21} = I_{22}I_{21} = 0. \tag{19}$$

We have to prove, that $E = LAU$. This equation in block form consists of four block equalities:

$$\begin{array}{l} E_{11} = L_{12}L_{11}A_{11}U_{11}U_{21}; \\ E_{12} = L_{12}L_{11}(A_{12}U_{12} - A_{11}U_{11}V)U_{22}; \\ E_{21} = L_{22}(L_{21}A_{21} - WL_{11}A_{11})U_{11}U_{21}; \\ E_{22} = L_{22}((L_{21}A_{22} - WL_{11}A_{12})U_{12} - (L_{21}A_{21} - WL_{11}A_{11})U_{11}V)U_{22}. \end{array} \tag{20}$$

Therefore we have to prove these block equalities.

Let us note, that from the identity $A_{11} = I_1 A_{11} J_1$ and Sentence 1 we get

$$L_{11} = \overline{I}_{11} + I_1 L_{11} I_{11},\ U_{11} = \overline{J}_{11} + J_{11} U_{12} J_1. \tag{21}$$

The Sentence 1 together with equations $A^1_{12} = \overline{I}_{11} L_{11} A_{12}$, $A^1_{21} = A_{21} U_{11} \overline{J}_{11}$, $A^2_{22} = \overline{I}_{21} L_{21} (A_{22} - A_{21} U_{11} E^T_{11} L_{11} A_{12}) U_{12} \overline{J}_{12}$ give the next properties of L- and U- blocks:

$$\begin{array}{ll} L_{12} = \overline{I}_{12} + \overline{I}_{11} I_1 L_{12} I_{12},\ U_{12} = \overline{J}_{12} + J_{12} U_{12} J_2, \\ L_{21} = \overline{I}_{21} + I_2 L_{21} I_{21}, & U_{21} = \overline{J}_{21} + J_{21} U_{12} J_1 \overline{J}_{11}, \\ L_{22} = \overline{I}_{22} + \overline{I}_{21} I_2 L_{22} I_{22}, & U_{22} = \overline{J}_{22} + J_{22} U_{22} J_2 \overline{J}_{12}. \end{array} \tag{22}$$

The following identities can be easy checked now

$$L_{12}E_{11} = E_{11},\ L_{12}I_{11} = I_{11}, \tag{23}$$

$$E_{11}U_{21} = E_{11},\ J_{11}U_{21} = J_{11}, \tag{24}$$

$$E_{12}U_{22} = E_{12},\ J_{12}U_{22} = J_{12}, \tag{25}$$

$$L_{22}E_{21} = E_{21},\ L_{22}I_{21} = I_{21}. \tag{26}$$

We shall use the following equalities,

$$L_{11}A_{11}U_{11} = E_{11}, L_{12}A^1_{12}U_{12} = E_{12}, L_{21}A^1_{21}U_{21} = E_{21}, L_{22}A^2_{22}U_{22} = E_{22}, \tag{27}$$

which follows from (7),(10) and (12), the equality

$$E_{11}V = I_{11}QU_{12}, \tag{28}$$

which follows from the definition of the block V in (13), (24), (16) and (6), the equality

$$WE_{11} = L_{21}BJ_{11}, \tag{29}$$

which follows from the definition of the block W in (13), (23), (17) and (6).

1. The first equality of (20) follows from (27), (23) and (24).

2.The right part of the second equality of (20) takes the form $L_{12}(\mathbf{I} - I_{11})QU_{12}U_{22}$ due to (8), (27) and (28). To prove the second equality we use the definition of the blocks B and A^1_{12} in (8) and (9), then the second equality in (27) and identity (25): $L_{12}(\mathbf{I} - I_{11})QU_{12}U_{22} = L_{12}A^1_{12}U_{12}U_{22} = E_{12}U_{22} = E_{12}$.

3. The right part of the third equality of (20) takes the form $L_{22}L_{21}B(\mathbf{I} - J_{11})U_{21}$ due to definition of the block B (8), the first equality in (27) and (29). To prove the third equality

we use the definition of the blocks A^1_{21} in (9), then the third equality in (27) and identity (26): $L_{22}L_{21}B\overline{J}_{11}U_{21} = L_{22}L_{21}A^1_{21}U_{21} = L_{22}E_{21} = E_{21}$.

4. The identity

$$E^T_{12}L_{12} = E^T_{12}L_{12}(I_{11} + \overline{I}_{11}) = E^T_{12}L_{12}\overline{I}_{11} \tag{30}$$

follows from (23) and (17).

We have to check that $(L_{21}A_{22} - WL_{11}A_{12})U_{12} = (L_{21}A_{22} - (GE^T_{12}L_{12} + L_{21}BE^T_{11})Q)U_{12} = L_{21}(A_{22} - BE^T_{11}Q)U_{12} - GE^T_{12}L_{12}QU_{12} = L_{21}A^1_{22}U_{12} - GE^T_{12}L_{12}\overline{I}_{11}QU_{12} = G - GE^T_{12}L_{12}A^1_{12}U_{12} = G - GE^T_{12}E_{12} = G\overline{J}_{12}$, using the definitions of the blocks W in (13), A^1_{22} and A^1_{12} in (9), the identity (28), the second equality in (27) and the definition (6).

We have to check that $-(L_{21}A_{21} - WL_{11}A_{11})U_{11}V = -(L_{21}A_{21}U_{11} - WE_{11})V = (-L_{21}B + L_{21}BJ_{11})V = -L_{21}B\overline{J}_{11}V = -L_{21}B\overline{J}_{11}(U_{21}E^T_{21}G\overline{J}_{12} + E^T_{11}QU_{12}) = -L_{21}A^1_{21}U_{21}E^T_{21}G\overline{J}_{12} = -I_{21}G\overline{J}_{12}$, using the first equality in (27), the identity (29), the definitions of the blocks V in (13), (1), then the third equality in (27) and definition (6).

To prove the forth equality we have to substitute obtained expressions to the right part of the fourth equality:

$$L_{22}(G\overline{J}_{12} - I_{21}G\overline{J}_{12})U_{22} = L_{22}\overline{I}_{21}G\overline{J}_{12}U_{22} = L_{22}A^2_{22}U_{22} = E_{22}.$$

For the completion of the proving of this theorem we have to demonstrate the special form of the matrices U and L: $L - \overline{I}_E \in F_{I,I_E}$ and $U - \overline{J}_E \in F_{J_E,J}$.

The matrix L is invertible and $I_E < I$ therefore we have to prove that $L = \overline{I}_E + ILI_E$, where $I_E = \mathrm{diag}(I_{11} + I_{12}, I_{21} + I_{22})$, $\overline{I}_E = \mathrm{diag}(\overline{I}_{11}\overline{I}_{12}, \overline{I}_{21}\overline{I}_{22})$, $I = \mathrm{diag}(I_1, I_2)$.

This matrix equality for matrix L (14) is equivalent to the four block equalities:

$$L_{12}L_{11} = I_1L_{12}L_{11}(I_{11} + I_{12}) + \overline{I}_{11}\overline{I}_{12},\ 0 = I_10(I_{21} + I_{22}),$$

$$-L_{22}WL_{11} = -I_2L_{22}WL_{11}(I_{11} + I_{12}),\ L_{22}L_{21} = I_2L_{22}L_{21}(I_{21} + I_{22}) + \overline{I}_{21}\overline{I}_{22}.$$

To prove the first block equalities we have to multiply its left part by the unit matrix in the form $\mathbf{I} = (I_1 + \overline{I}_1)$ from the left side and by the unit matrix in the form $\mathbf{I} = (I_{11} + I_{12}) + \overline{I}_{11}\overline{I}_{12}$ from the left side. Then we use the following identities to obtain in the left part the same expression as in the right part: $L_{11}\overline{I}_{11} = \overline{I}_{11}$, $L_{12}\overline{I}_{12} = \overline{I}_{12}$, $\overline{I}_1L_{12}L_{11} = \overline{I}_1$, $\overline{I}_1(I_{11} + I_{12}) = 0$. The same idea may be used for proving the last block equality, but we must use other forms of unit matrix: $\mathbf{I} = (I_2 + \overline{I}_2)$, $\mathbf{I} = (I_{21} + I_{22}) + \overline{I}_{21}\overline{I}_{22}$.

The second block equality is evident.

Let us prove the third block equality. We have to multiply the left part of the third block equality by the unit matrix in the form $\mathbf{I} = (I_2 + \overline{I}_2)$ from the left side and by the unit matrix in the form $\mathbf{I} = (I_{11} + I_{12}) + \overline{I}_{11}\overline{I}_{12}$ from the right side.

The block W is equal to the following expression by the definition (13), (11) and (8):

$$W = (L_{21}(A_{22} - A_{21}U_{11}E^T_{11}Q)U_{12}E^T_{12}L_{12} + L_{21}A_{21}U_{11}E^T_{11}).$$

We have to use in the left part the equations $\overline{I}_2L_{22} = \overline{I}_2$, $\overline{I}_2L_{21} = \overline{I}_2$, $\overline{I}_2A_{22} = 0$, $\overline{I}_2A_{21} = 0$, and $L_{11}\overline{I}_{11} = \overline{I}_{11}$, $L_{12}\overline{I}_{12} = \overline{I}_{12}$, $E^T_{12}\overline{I}_{12} = 0$, $E^T_{11}\overline{I}_{11} = 0$.

The property of the matrix U: $U - \overline{J}_E \in F_{J_E,J}$ may be proved in the same way as the property of the matrix L.

References

1. *Grigoriev D.* Analogy of Bruhat decomposition for the closure of a cone of Chevalley group of a classical serie // Soviet Math. Dokl.1981. V. 23. N. 2. P. 393-397.

2. *Bunch J., Hopkroft J.* Triangular factorization and inversion by fast matrix multiplication // Mat. Comp. 1974. V. 28. P. 231-236.

3. *Malaschonok G.I.* Effective Matrix Methods in Commutative Domains // Formal Power Series and Algebraic Combinatorics. Berlin: Springer, 2000. P. 506-517.

4. *Malaschonok G.I.* Matrix computational methods in commutative rings. Tambov: Tambov State University, 2002.

5. *Akritas A., Malaschonok G.* Computation of Adjoint Matrix // Fourth International Workshop on Computer Algebra Systems and Applications (CASA 2006), LNCS 3992. Berlin: Springer, 2006. P. 486-489.

6. *Watt S.M.* Pivot-Free Block Matrix Inversion. Maple Conference 2006, July 23-26, Waterloo, Canada. 2006. URL: http://www.csd.uwo.ca/ watt/pub/reprints/2006-mc-bminv-poster.pdf.

7. *Watt S.M.* Pivot-Free Block Matrix Inversion // Proc 8th International Symposium on Symbolic and Numeric Algorithms in Symbolic Computation (SYNASC), IEEE Computer Society. 2006. P. 151-155.

8. *Eberly W.* Efficient parallel independent subsets and matrix factorization // 3rd IEEE Symposium on Parallel and Distributed Processing. Dallas, USA, 1991. P. 204-211.

9. *Kaltofen E., Pan V.* Processor-efficient parallel solution of linear systems over an abstract field // 3rd Annual ACM Symposium on Parallel Algorithms and Architectures. ACM Press, 1991. P. 180-191.

10. *Kaltofen E., Pan V.* Processor-efficient parallel solution of linear systems II: The general case // 33rd IEEE Symposium on Foundations of Computer Science. Pittsburgh, USA, 1992. P. 714-723.

11. *Kaltofen E., Pan V.* Parallel solution of Toeplitz and Toeplitz-like linear systems over fields of small positive characteristic // PASCO 94: First International Symposium on Parallel Symbolic Computation, World Scientific Publishing, 1994. P. 225-233.

12. *Grigoriev D.* Additive complexity in directed computations // Theoretical Computer Science. 1982. V. 19. P. 39-67.

13. *Kolotilina L.Yu.* Sparsity of Bruhat decomposition factors of nonsingular matrices // Notes of Scientific Seminars of LOMI. 1992. V. 202. P. 5-17.

14. *Kolotilina L.Yiu, Yeremin A.Yu.* Bruhat decomposition and solution of linear algebraic systems with sparse matrices // Sov. J. Numer.Anal. and Math. Model. 1987. V. 2. P. 421-436.

15. *Malaschonok G.I.* Parallel Algorithms of Computer Algebra // Materials of the conference dedicated for the 75 years of the Mathematical and Physical Dep. of Tambov State University. (November 22-24, 2005). Tambov: TSU, 2005. P. 44-56.

16. *Malaschonok G.I., Zuyev M.S.* Generalized algorithm for computing of inverse matrix // 11-th conference "Derzhavinskie Chtenia". February 2-6, 2006. Tambov: TSU, 2006. P. 58-62.

17. *Malaschonok G.I.* On computation of kernel of operator acting in a module // Tambov University Reports. Natural and Technical Sciences. 2008. V. 13. Issue 1. P. 129-131.

18. *Strassen V.* Gaussian Elimination is not optimal // Numerische Mathematik. 1969. V. 13. P. 354-356.

GRATITUDES: Supported by the Sci. Program Devel. Sci. Potent. High. School, RNP 2.1.1.1853.

Accepted for edition 7.06.2010.

БЫСТРОЕ МАТРИЧНОЕ РАЗЛОЖЕНИЕ В ПАРАЛЛЕЛЬНОЙ КОМПЬЮТЕРНОЙ АЛГЕБРЕ

© **Геннадий Иванович Малашонок**

Тамбовский государственный университет им. Г.Р. Державина, Интернациональная, 33, Тамбов, 392000, Россия, доктор физико-математических наук, профессор кафедры математического анализа, e-mail: malaschonok@ya.ru

Ключевые слова: быстрые алгоритмы; матричное разложение; параллельные алгоритмы; компьютерная алгебра.

Предложены новые алгоритмы для вычисления матричного разложения и для вычисления обратной матрицы в случае матриц над произвольными полями. Для коммутативных областей предложен алгоритм вычисления присоединенной матрицы. Эти алгоритмы имеют сложность матричного умножения и не требуют поиска ведущего элемента и выполнения перестановок элементов матриц. Для вырожденных матриц они позволяют находить невырожденный блок наибольшего размера. Предлагаемые алгоритмы не требуют пилотирования и не меняют матричную блочную структуру. Эти алгоритмы позволяют разрабатывать соответствующие параллельные программы.

UDC 519.688

A PARALLEL ALGORITHM FOR SYMBOLIC SOLVING PARTIAL DIFFERENTIAL EQUATIONS

© **Natalia Aleksandrovna Malaschonok**
Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, Candidate of Physics and Mathematics, Associate Professor of Mathematical Analysis Department, e-mail: namalaschonok@gmail.com

Key words: parallel algorithms; computer algebra; partial differential equations; Laplace–Carson transform; compatibility conditions.

A parallel algorithm for symbolic solving partial differential equations by means of Laplace–Carson transform is produced. The problem is reduced to solving linear algebraic systems with polynomial coefficients, for which efficient parallel algorithms exist. It permits to construct a fast parallel algorithm for systems of partial differential equations. An algorithm includes a procedure to obtain compatibility conditions for initial data.

1 Introduction

An application of Laplace and Laplace–Carson transform is useful in many problems of solving differential equations (for example [1, 2, 3, 4]) It reduces a system of partial differential equations to an algebraic linear system with polynomial coefficients. Parallel algorithms for solving such systems are being developed actively (for example, [5, 6]). It enables to construct parallel algorithms for solving linear partial differential equations with constant coefficients and systems of equations of various order, size and types. The application of Laplace–Carson transform permits to obtain compatibility conditions in symbolic way for many types of PDE equations and systems of PDE equations.

The steps, at which parallel calculations are possible and reasonable we denote by term **Block**. If indexes are contained, the ways of parallelization are pointed by them.

2 Input data

Denote $\widetilde{m} = (m_1, \ldots, m_n)$. Consider a system

$$\sum_{k=1}^{K} \sum_{m=0}^{M} \sum_{\widetilde{m}} a^j_{\widetilde{m}k} \frac{\partial^m}{\partial^{m_1} x_1 \ldots \partial^{m_n} x_n} u_k(x) = f_j(x), \tag{1}$$

where $j = 1, \ldots, K$, $u_k(x)$, $k = 1, \ldots, K$, — are unknown functions of $x = (x_1, \ldots, x_n) \in \mathbf{R}^n_+$, $f_j \in S$, $a^j_{\widetilde{m}k}$ are real numbers, m is the order of a derivative, and k –the number of an unknown function. Here and further summing by $\widetilde{m} = (m_1, \ldots, m_n)$ is executed for $m_1 + \ldots + m_n = m$.

We consider all input functions reducible to the form;

$f_j(t) = f^i_j(x)$, $x^i_j < t < t^{i+1}_j$, $i = 1, \ldots, I_j, x^1_l = 0, t^{I_j+1}_j = \infty$,

where

$$f_j^i(t) = \sum_{s=1}^{S_j^i} P_{js}^i(t) e^{b_{js}^i t}, \quad i = 1, \ldots, I_j, \quad j = 1, \ldots, k, \tag{2}$$

and $P_{js}^i(x) = \sum_{l=0}^{L_{js}^i} c_{sl}^{ji} x^l$.

Denote by $\mathbf{A}$ a class of functions which are reducible to the form (2).

We solve a problem with initial conditions for each variable. Introduce notations for them. Denote by Γ^ν a set of vectors $\gamma = (\gamma_1, \ldots, \gamma_n)$ such that $\gamma_\nu = 1$, $\gamma_i = 0$, if $i < \nu$, and γ_i equals 0 or 1 in all possible combinations for $i > \nu$. The number of elements in Γ^ν equals $2^{\nu-1}$.

Denote $\beta = (\beta_1, \ldots, \beta_n)$, $\beta_i = 0, \ldots, m_i$, a set of indexes such that the derivative of $u^k(x)$ of the order β_i with respect to the variables with numbers i equals $u_{\beta,\gamma}^k(x^{(\gamma)})$ at the point $x = x^\gamma$ with zeros at the positions μ for which the coordinates γ_μ of γ equal 1. For example, if zeros stand only at the places with the numbers $1, 2, 3$, then $\gamma = (1, 1, 1, 0, \ldots, 0)$. Functions $u_{\beta,\gamma}^k(x^{(\gamma)})$ must also belong to $\mathbf{A}$. To be short we shall not write down the expressions for $u_{\beta,\Gamma}^k(x^{(\gamma)})$.

The algorithm component is the definition of compatible initial conditions. The system (1) is to be solved under such conditions.

Data file contains the coefficients, the initial conditions and the right-hand members f_j, $l = 1, \ldots, K$.

The data for functions f_j consists of the polynomial coefficients, parameters of exponents, the bounds of smoothness intervals.

3 Laplace–Carson transform

Consider the space S of functions $f(x)$, $x = (x_1, \ldots, x_n) \in \mathbf{R}_+^n$, $\mathbf{R}_+^n = \{x : x_i \geqslant 0, i = 1, \ldots, n\}$, for which $\mathcal{M} > 0, a = (a_1, \ldots, a_n) \in \mathbf{R}^n$, $a_i > 0$, $i = 1, \ldots, n$, exist such that for all $x \in \mathbf{R}_+^n$ the following is true: $|f(x)| \leqslant \mathcal{M} e^{ax}$, $ax = \sum_{i=1}^n a_i x_i$.

On the space S the Laplace–Carson transform (**LC**) is defined as follows:

$$LC : f(x) \mapsto F(p) = p^1 \int_0^\infty e^{-px} f(x) dx,$$

$$p = (p_1, \ldots, p_n), \quad p^1 = p_1 \ldots p_n,$$

$$px = \sum_{i=1}^n p_i x_i, \quad dx = dx_1 \ldots dx_n.$$

LC is performed symbolically at the class $\mathbf{A}$.

4 Parallel LC algorithm

4.1 LC of a system

Let $LC : u^k \mapsto U^k, u_{\beta,\gamma}^k(x^{(\gamma)}) \mapsto U_{\beta,\gamma}^k(p^{(\gamma)}), f_j \mapsto F_j$, the notation $p^{(\gamma)}$ is correspondent to the notation $x^{(\gamma)}$. Denote by $\|\gamma\|$ the "length" of γ — the number of units in γ, $p^{\widetilde{m}} = p_1^{m_1} \ldots p_n^{m_n}$.

Block 10

The LC of the left-hand side of the system (1) excluding images of initial conditions is written formally.

Block 1r

r runs trough the set of multiindexes of $u^k_{\beta,\Gamma}(x^\Gamma)$.

Then

$$LC : \frac{\partial^m}{\partial^{m_1} x_1 \dots \partial^{m_n} x_n} u_k(x) \ \mapsto$$

$$p^{\widetilde{m}} U^k(p) \ + \ \sum_{\nu=1}^{n} \sum_{\beta_\nu=0}^{m_\nu} \sum_{\gamma \in \Gamma^\nu} (-1)^{\|\gamma\|} p_1^{m_1-\beta_1-\gamma_1} \dots p_n^{m_n-\beta_n-\gamma_n} U^k_{\beta,\gamma}(p^{(\gamma)}).$$

Denote

$$\Phi^j_{mk} = \sum_{\widetilde{m}} a^j_{\widetilde{m}k} \sum_{\nu=1}^{n} \sum_{\beta_\nu=0}^{m_\nu} \sum_{\gamma \in \Gamma^\nu} (-1)^{\|\gamma\|} p_1^{m_1-\beta_1-\gamma_1} \dots p_n^{m_n-\beta_n-\gamma_n} U^k_{\beta,\gamma}(p^{(\gamma)}).$$

As a result of Laplace–Carson transform of the system (1) according to initial conditions we obtain an algebraic system relative to U^k

$$\sum_{k=1}^{K} \sum_{m=0}^{M} \sum_{\widetilde{m}} a^j_{\widetilde{m}k} p^{\widetilde{m}} U^k(p) = F_j - \sum_{k=1}^{K} \sum_{m=0}^{M} \Phi^j_{mk}, j = 1, \dots, K. \tag{3}$$

Block 2k

k runs from 1 to K.

These blocks performs LC of the right-hand parts of (1). **A** allows a further parallelization of calculations.

4.2 Solution of algebraic system

Block 3

As a result of Laplace–Carson transform of the system (1) according to initial conditions we obtain the algebraic system (3) relative to U^k.

Efficient methods of parallel solving such systems are developed (for example [5, 6]).

At this stage the problem of definition of compatibility conditions arises (see blocks 4s,5). With respect to compatible conditions we use the inverse Laplace–Carson transform and obtain the correct solution of PDE system.

4.3 Compatibility conditions

Call a rational fraction *"a proper fraction"* if the degree of each variable (over **C**) in numerator is less then its degree in denominator.

Call a set of equations, defined by conditions

- the solutions of algebraic system may be represented as sums of proper fractions with exponential coefficients;

• the denominators of these proper fractions may be reduced to a product of linear functions.
the class **B**.

(Note that the class **B** does not exhaust all cases that admit pure symbolic computations.)

Denote by D the determinant of the system (3), D_i the maximal order minors of the extended matrix of (3). A case when there is a set $\mathcal{Q}$ of zeros of D with infinite limit point at $\operatorname{Re} p_k > 0$, $k = 1, \ldots, n$, is of most interest. Solving the system (1) we obtain U^k as fractions with D in the denominators. The inverse Laplace–Carson transform is possible if α_k, $k = 1, \ldots, n$, exist such that these functions are holomorphic in the domain $\operatorname{Re} p_k > \alpha_k$. So we make a demand: D_i has zeros at $\mathcal{Q}$ of multiplicity not less than multiplicity of corresponding zeros of D. This demand produces requirements to the LC images of initial conditions functions, and after LC^{-1} transform – to initial conditions. They turn to be dependent. We obtain the so-called compatibility conditions.

Block 4s

s depends upon the number of relations, from which the compatibility conditions arise.

The blocks calculate the values of numerators at zeros of denominators.

Block 5

The block implements parallel solving of the system of equations, produced by relations for compatibility conditions.

Block 6k

The blocks perform the LC^{-1} of U^k. Note, that the steps of calculation of multivariate LC^{-1} are produced sequentially.

5 Example

We take a simple example to demonstrate the method and the places where parallelization is possible.

It is convenient here to change notations for unknown functions, their Laplace transform, variables, initial conditions.

Example 1

Take a system of two equations with two unknown functions on $\mathbf{R}^2_+$.

$$\begin{cases} \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} = x, \\ \frac{\partial f}{\partial y} + \frac{\partial g}{\partial x} = y, \end{cases}$$

$f = f(x, y); \quad g = g(x, y)$.

Initial conditions: $f(0, y) = a(y); \quad f(x, 0) = b(x); \quad g(0, y) = c(y); \quad g(x, 0) = d(x),$

Block 1r , r=1,2.

$$a(y) \mapsto \alpha(q), \quad b(x) \mapsto \beta(p),$$
$$c(y) \mapsto \delta(q), \quad d(x) \mapsto \gamma(p).$$

Block 2k , k=1,2.
LC:

$$f(x,y) \mapsto u(p,q), \quad g(x,y) \mapsto v(p,q).$$

As a result of LC we obtain the algebraic system:

$$pu - p\alpha(q) + qv - q\gamma(p) = 1/p, \quad qu - q\beta(p) + pv - p\delta(q) = 1/q.$$

Block 3
Then

$$u = -\frac{-\alpha p^2 + \beta q^2 + (\delta - \gamma)pq}{p^2 - q^2}, \quad v = -\frac{-p^2 + q^2 + (\alpha - \beta)p^2q^2 - (\delta p^2 - \gamma q^2)pq}{pq(p^2 - q^2)}.$$

The denominator D: $D(p,q) = pq(p^2 - q^2)$.

Block 4s , s=1.

The set of zeros of D with infinite limit points at the right half-plane is $q = p$.

Substituting $q = p$ into the nominator of u and v we obtain the compatibility condition: $\alpha - \beta + \gamma - \delta = 0$.

Block 5

For example we may take $\beta = 0; \quad \gamma = \frac{2}{p}; \quad \delta = \frac{2}{q}; \quad \alpha = 0.$
Then

$$u = -\frac{2}{p+q}, \quad v = -\frac{p + 2p^2 + q + 2q^2 + 2pq}{pq(p+q)}.$$

Block 6s , s=1,2.
LC^{-1} :

$$f = -\begin{cases} 2y, & y < x, \\ 2x, & y \geqslant x, \end{cases}$$
$$g = \begin{cases} (2+y)x, & y < x, \\ y(2+x), & y \geqslant x. \end{cases}$$

References

1. *Dahiya R.S., Jabar Saberi-Nadjafi.* Theorems on n-dimensional Laplace transforms and their applications // 15th Annual Conf. of Applied Math., Univ. of Central Oklahoma, Electr. Journ. of Differential Equations, Conf.02. 1999. P. 61-74.

2. *Dimovski I., Spiridonova M.* Computational approach to nonlocal boundary value problems by multivariate operational calculus // Mathem. Sciences Research Journal. Dec. 2005. V. 9. N. 12. P. 315-329.

3. *Malaschonok N.* An algorithm for symbolic solving of differential equations and estimation of accuracy // Computer Algebra in Scientific Computing. CASC 2009, Springer-Verlag Berlin Heidelberg, 2009. P. 213-225.

4. *Picone M.* Nuovi metodi risolutivi per i problemi d'integrazione delle equazioni lineari a derivate parziali e nuova applicazionne della trasformata multipla di Laplace nel caso delle equazioni a coefficienti costanti // Atti Accad.Sci.Torino,75. 1940. P. 1-14.

5. *Watt S.M.* Pivot-Free Block Matrix Inversion, Proc 8th International Symposium on Symbolic and Numeric Algorithms in Symbolic Computation (SYNASC), IEEE Computer Society, 2006. P. 151-155. URL: http://www.csd.uwo.ca/ watt/pub/reprints/2006-synasc-bminv.pdf.

6. *Malaschonok G.I.* Parallel Algorithms of Computer Algebra // Materials of the conference dedicated for the 75 years of the Mathematical and Physical Dep. of Tambov State University. (November 22-24, 2005). Tambov: TSU, 2005. P. 44-56.

GRATITUDES: Supported by the Sci. Program Devel. Sci. Potent. High. School, RNP 2.1.1.1853.

Accepted for publication 7.06.2010.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ СИМВОЛЬНОГО РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ С ЧАСТНЫМИ ПРОИЗВОДНЫМИ

© **Наталия Александровна Малашонок**

Тамбовский государственный университет им. Г.Р. Державина, Интернациональная, 33, Тамбов, 392000, Россия, кандидат физико-математических наук, доцент кафедры математического анализа, e-mail: malaschonok@ya.ru

Ключевые слова: параллельные алгоритмы, компьютерная алгебра, уравнения в частных производных, преобразование Лапласа–Карсона, условия согласованности.

Представлен параллельный алгоритм символьного решения системы уравнений с частными производными с помощью преобразования Лапласа–Карсона. Задача сводится к решению линейной алгебраической системы с полиномиальными коэффициентами, для которой существуют быстрые параллельные алгоритмы. это позволяет сконструировать быстрый параллельный алгоритм для систем дифференциальных уравнений с частными производными. Составной частью алгоритма является процедура получения условий согласованности для начальных условий.

UDC 5519.852+681.142

EXACT AND GUARANTEED ACCURACY SOLUTIONS OF LINEAR PROGRAMMING PROBLEMS BY DISTRIBUTED COMPUTER SYSTEMS WITH MPI

© **Anatoliy Vasilyevich Panyukov**
South Ural State University, 76 Lenina Ave., Chelyabinsk, 454080, Russia, Doctor of Physics and Mathematics, Professor, Head of Economical and Mathematical Methods and Statistics Department, e-mail: a_panykov@mail.ru

© **Vasiliy Vladimirovich Gorbik**
South Ural State University, 76 Lenina Ave., Chelyabinsk, 454080, Russia, Post-graduate Student of Economical and Mathematical Methods and Statistics Department, e-mail: gorbik@gmail.com

Key words: linear programming; tabular simplex method; distributed computing; parallel optimization; rational computations; arbitrary precision; interval arithmetic.

Techniques of obtaining both exact and guaranteed accuracy solutions of linear programming problems and methods of increasing accuracy of computations by distributed computer systems with MPI are subjects of this paper. To obtain the solutions the rational and arbitrary precision floating point interval arithmetic libraries are applied. Methods of adaptation of the used data types to MPI are presented. Results of computational experiments based on introduced parallel versions of algorithms for solving systems of linear equations and linear programming problems demonstrate effectiveness of their application.

1 Introduction

Unsubstantiated prejudices, causing errors of calculations are widespread. Some of them are: (1) distributing property of associativity of addition and multiplication in the field of real numbers to a finite set of machine "real" numbers; (2) extension of properties of continuous dependence on parameters of solutions of the system received after the "equivalent" changes to the original system. Calculations that ascribe non-existing properties to objects of the numerical analysis, are unproved. Popular commercial packages MatLab, MathCad, etc., and also free package SciLab have the marked disadvantages. Usage of different number of processors in calculations in many cases gives substantially different results, demonstrating the need for evidence-based computing. The potential of available packages supporting symbolic computations does not allow to solve the real problems of mathematical modeling. For the means of arbitrary precision computations GMP library can be used. But GMP library does not provide any interface for using it in parallel computations.

The aim of this work is implementation of exact rational and guaranteed arbitrary precision floating point interval computations software for parallel and distributed computing systems with MPI (Message Passing Interface[1]). The paper covers the usage of mpq_t and mpf_t

data types from the GNU MP library [2], and mpfi_t interval type from MPFI library [3](that is built on the top of GNU MP). An important aspect here is the possibility and effectiveness of adaptation of these types to a multiprocessor environment. MPI interface is an unofficial standard for building distributed computing systems for a long time. Serialization and rearrangement to sequential memory layout of rational and interval arithmetic objects for MPI integration are considered in this paper. We chose GNU MP library for the purposes of exact computations because it is an open source solution available in all widespread GNU/Linux distributions and has a good performance. MPFI library is also an open source project and extends GNU MP; adding interval calculations on top of arbitrary precision floating point data types.

2 Accuracy of Computations

The GNU MP package contains open source GNU MP library for arbitrary precision arithmetic: operations on signed integers, rational numbers and floating-point numbers. GNU MP library is developed for fast operation on both large and small operands. It is fast because it uses whole words as the base type, applies fast algorithms, depending on the size of the operands. It has the optimized assembly code for many types of processors and combines speed with simplicity and elegance of operations.

2.1 Exact Computations with mpq_t Type

```
//   FILE: gmp.h
//Definition of mpq_t
#ifdef __GMP_SHORT_LIMB
  typedef unsigned int mp_limb_t;
#else
#ifdef _LONG_LONG_LIMB
  typedef unsigned long \
    long int mp_limb_t;
#else
typedef unsigned long \
  int  mp_limb_t;
#endif
#endif
```

```
typedef struct
{
  int _mp_alloc;
  int _mp_size;
  mp_limb_t *_mp_d;
} __mpz_struct;

typedef struct
{
  __mpz_struct _mp_num;
  __mpz_struct _mp_den;
} __mpq_struct;

typedef __mpq_struct mpq_t[1];
```

Fig. 1. Declaration of mpq_t type

By the means of mpq_t type exact calculations with rationals are implemented, mpq_t type is based on structures and functions of C library, numerator and a denominator are mpz_struct structures which contain:

- size of allocated memory;
- size of occupied memory;
- pointer to an array representing number.

Code fragment on figure 1 shows mpq_t type declaration.

GNU MP library contains about 40 functions for mpq_t type. Besides it is possible to apply any integers functions to its numerator and denominator separately.

2.1.1 Adaptation of mpq_t Type to MPI

Effective transmission of mpq_t type for MPI environment can be carried out by the means of incomplete serialization. Details of implementation and efficiency estimation are presented in the previous papers [4], [5].

2.2 Arbitrary Precision Floating Point and Interval Computation

In a case when problem has such a scale that it is impossible to use exact computations with rational types, and inaccuracy of solution with hardware floating-point data types fall outside of admissible limits we can use arbitrary precision floating point data types. Effective implementations of such derivative data types, unlike rational, are comparable to the computation time with hardware floating point (with similar length of a mantissa). One of such data types is mpf_t (multiple precision floating-point). It allows dynamically arbitrary change accuracy (the length of mantissa). But computations with mpf_t data type are approximate, and inaccuracy is not considered. Data type mpf_t can be used in algorithms when result verification procedure exists allowing to measure inaccuracy and repeat algorithm from some step with more precision if necessary.

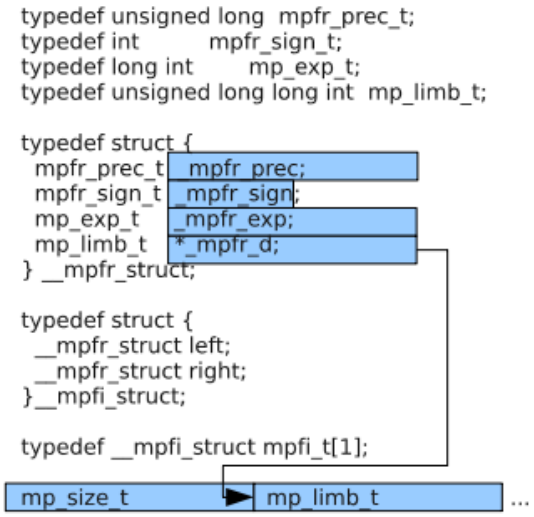

Fig. 2. Structure of mpfi_t type

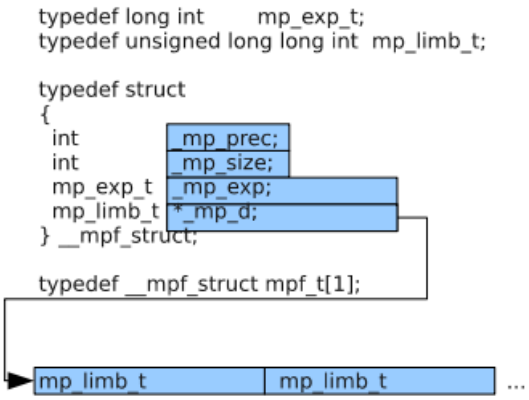

Fig. 3. Structure of mpf_t type

For the means of guaranteed solution one can use data mpfi_t type (from the multiple precision floating-point interval library [3]). The mpfi_t library is based on GNU MP and mpfr (multiple-precision floating point with correct rounding library [6]). Instead of single floating point a number in mpfi is represented by a pair of arbitrary precision floating point values (of mpfr type), they represent lower and upper interval bounds enclosing real value. Unlike mpf, mpfi allows to obtain guaranteed (due to usage of interval calculations) and accurate results (due to arbitrary precision and correct rounding according standard IEEE 754, implemented in mpfr). On figures 2 and 3 structures of data types mpfi_t and mpf_t (for 64-bit architectures) are shown.

2.2.1 Adaptation of mpfi_t and mpf_t Data Types to MPI

The complexity of effective MPI adaptation of derived types with dynamic sizes exist because MPI doesn't provide any interface for passing data types that (1) don't have sequential memory layout or (2) have variable length that could be changed multiple times at runtime. Mantissas of mpfi_t and mpf_t data types are stored out of base structure. In mpf_t structure field _mp_d points to mantissa. In a case of mpfi_t field _mpfr_d pointers of _mpfr structures points to the second elements of the allocated memory blocks (the beginning of mantissa), the current length of mantissa is stored in the first element. Thus, the data can have random memory locations, and it is impossible to define MPI data type on top of mpfi_t and mpf_t types. But still we can perform effective transmission in two ways:

- incomplete serialization;
- rearrangement to sequential memory layout.

In case of incomplete serialization it is enough packing of the necessary structure fields that have been painted over at figures 4 and 5.

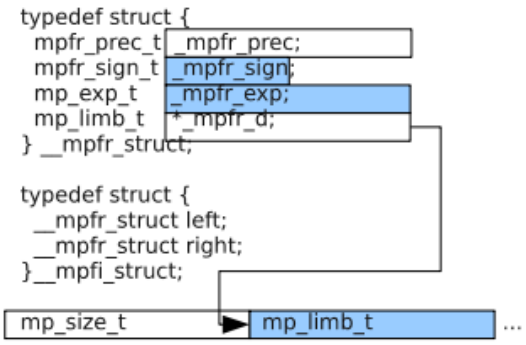

Fig. 4. Serialization of mpfi_t type

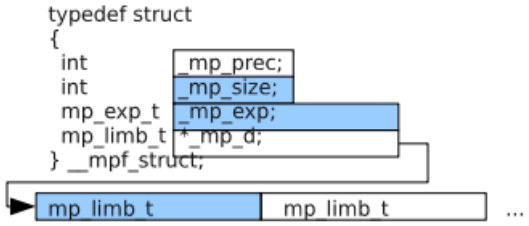

Fig. 5. Serialization of mpf_t type

Depending on the algorithm, if the receiver is not aware of the data types precision of the sender, _mpfr_prec, _mp_prec fields should be also serialized and transfered. In some cases, only _mp_size elements of the mantissa array may be transfered (in the general case _mp_size is equal to the total length _mp_prec).

This approach has obvious disadvantage, that in fact we have to implement the MPI transfer functions by transferring array of bytes (in case of incomplete mantissa transfer - array of bytes previously unknown size), and in case of collective communication it's not always an easy task itself.

2.2.2 Memory Layout Rearrangement for mpfi_t and mpf_t

This approach makes sense in the case when the algorithm operates on the numbers with precision, that is not changed during the computations. Using the macros shown on figure 6,we can define the derived types mpfin_t and mpfn_t based on mpfi_t and mpf_t.

Obviously, the structure of types mpfin_t and mpfn_t, as well as arrays of objects of these types will be allocated in memory sequentially (not including alignment). In this case, all operations (except for initialization and precision set) that are available for mpfi_t (mpf_t), may be applied to mpfin_t (mpfn_t) without any restrictions. Initialization procedure is not

```
#define mpfi_type(prec) \
typedef struct \
{ \
  __mpfr_struct left; \
  __mpfr_struct right; \
  mp_limb_t \
    _mp_d_left[L(prec)+ 1]; \
  mp_limb_t \
    _mp_d_right[L(prec)+1]; \
} __mpfi_struct##prec; \
typedef __mpfi_struct##prec \
mpfi##prec##_t[1]; \
MPI_Datatype MPI_mpfi##prec;
```

```
#define mpf_type(prec) \
typedef struct \
{ \
  int _mp_prec; \
  int _mp_size; \
  mp_exp_t _mp_exp; \
  mp_limb_t *_mp_d; \
  mp_limb_t \
    _mp_d_real[L(prec)+ 1]; \
} __mpf_struct##prec; \
typedef __mpf_struct##prec \
mpf##prec##_t[1]; \
MPI_Datatype MPI_mpf##prec;
```

Fig. 6. Declaration of mpfin_t and mpfn_t types macros

fundamentally different from the original. Instead of memory allocation for mantissa simple initialization of pointers left._mpfr_d and right._mpfr_d (_mp_d) with relevant addresses of mantissa is required (which offset is now constant).

The significant positive factor of this approach is possibility easily declare the MPI data type on top of mpfin_t and mpfn_t types (for any given accuracy), and use all functions available for MPI-derived data types without any restrictions.

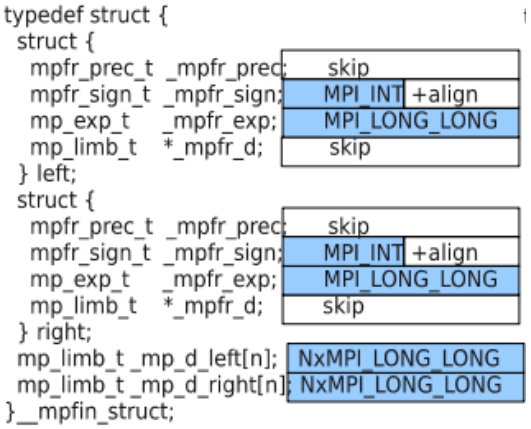

Fig. 7. Fields of mpfi_t structure included in MPI data type

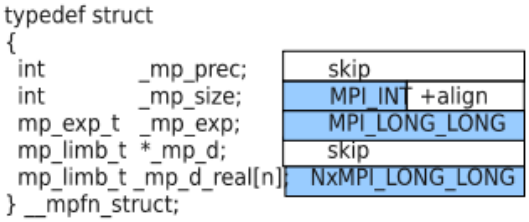

Fig. 8. Fields of mpf_t structure included in MPI data type

Fig. 7 and 8 show mpfin_t and mpfn_t data types structures in terms of MPI (for 64-bit architectures). If one declares MPI data type in a way as shown on fig. 7 and 8 skipping non-colored fields, the pointer to the mantissa won't be rewritten during data receive and no adjustments or additional steps need to be carried out during transfer at all.

Another question is memory layout rearrangement effects for performance. Table 1 shows a comparison of cache misses for initial and modified types. The comparison was made on the processor with 2 MB second-level cache (32 KB first-level) by solving the system of lineal equations by Gauss-Jordan elimination (30 equations and 192 bit mantissa precision).

Table 1

Cache miss test for initial and modified types

	mpfi	**mpfin**	**mpf**	**mpfn**
I refs:	43366674	41850711	13762859	13601951
I1 misses:	23822	21566	10938	10205
L2i misses:	3456	3447	3236	3228
D refs:	18095665	17450618	5076607	5012224
D1 misses:	84427	65950	46662	38923
L2d misses:	13121	12126	9886	9646
L2 refs:	108249	87516	57600	49128
L2 misses:	16577	15573	13122	12874

Test shows that modified types has better cache-hit rate than original. But in closer look on specific functions it turns out that the cache hit rate is better only for functions like comparison, addition, subtraction, etc., but somewhat worse for multiplication and division. There is a slight superiority of the original data types under increasing problem size to 3000 equations and mantissa precision to 1024-2048 bit.

3 Parallel Simplex Method

Simplex-method application for real-world linear program problems keeps beyond comparison in spite of appearance of polynomial algorithms [7]. At present two techniques of simplex-method software engineering are in use:

- tabular simplex-method;
- inverse pivotal matrix method (revised simplex-method).

Preserve generality let us demonstrate its characteristic property with an example linear programming problem

$$\max\left\{c^T x \ : Ax = b \geqslant 0,\ x \geqslant 0\right\}. \tag{1}$$

3.1 Tabular simplex-method

On k-th iteration it re-counts the simplex table

$$\left|\begin{array}{c|c} z^{(k)} = -c^T + c^T_{B(k)} B(k)^{-1} A & c^T_{B(k)} B(k)^{-1} b \\ \hline S^{(k)} = B(k)^{-1} A & x^{(k)} = B(k)^{-1} b \end{array}\right|,$$

here $B(k)$ is the pivotal matrix containing matrix A columns related to the basic variables, $c_{B(k)}$ is criterion function coefficient vector related to the basic variables. At that right simplex table column contains vector $x^{(k)} = B(k)^{-1} b$ of the basic variable values, and the criterion function value $c^T_{B(k)} B(k)^{-1} b$ for current solution. The upper row contains vector $z^{(k)} = -c^T +$

$c^T_{B(k)}B(k)^{-1}$ of relative evaluation replacement for nonbasic variables. Test for optimality of the current basic solution is nonnegativity of vector z.

If optimality test no passed then there is nonbasic variable $x_i:\ z_i^{(k)} < 0$. Let us introduce set $L = \{l:\ S_{li}^{(k)} > 0\}$. If $L = \emptyset$ then the criterion function is unbounded. Otherwise incoming of x_i to basic variables leads to criterion function increment

$$\Delta_i = -\frac{x_{l^*}^{(k)} \cdot z_i^{(k)}}{S_{l^*i}^{(k)}}, \tag{2}$$

here

$$l^* = \arg\min_{l \in L} \frac{x_l^{(k)}}{S_{li}^{(k)}}$$

defines the variable outgoing from basic ones.

Conversion from k-th iteration simplex table to $(k+1)$-th one is realized by Gauss-Jordan elimination for i-th column of the current simplex table. Principal computation capacity is block S converting that requires $\Theta(mn)$ algebraic operations (m is number of rows, n is number of columns of matrix A).

3.2 Inverse pivotal matrix method

On k-th iteration it re-counts matrix $B(k)^{-1}$ that requires $\Theta(m^2)$ algebraic operations ($m < n$). At that for each iteration it is necessary

- to compute basic variables value $x^{(k)} = B(k)^{-1}b$ ($\Theta(m^2)$ algebraic operations);
- to compute dual variables value $y^T = c^T_{B(k)}B(k)^{-1}$ ($\Theta(m^2)$ algebraic operations);
- to check permissibility of the dual solution $c^T \leqslant y^T A$ (no more $O(mn)$ algebraic operations).

If the dual problem is impressible there is nonbasic variable $x_i:\ z_i^{(k)} = -c_i^T + y^T A_i < 0$. If set $L = \{l:\ S_{li}^{(k)} > 0\} = \emptyset$, where $S_i^{(k)} = B(k)^{-1}A_i$ then the criterion function is unbounded. Otherwise incoming of x_i to basic variables leads to criterion function increment defined by formula (2). So application of inverse pivotal matrix method is realized when

- n essentially surpass m;
- matrix A is sparse;
- it is required to solve both primal and dual problems.

3.3 Intercomparison of methods

In the case of tabular simplex method, columns decomposition is preferred due to calculations and communication specific (fig. 9). All the columns $1, 2, \ldots, n$ can be divided in equal proportions between the processes $K = 1, 2, \ldots, N$, the vector of basic variables and the criterion function value are sent to all processes and are processed independently. So the basic steps of one algorithm iteration are following:

Process K				
$S_{00} = Z$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 1}$	$z_{\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$z_{\lceil \frac{Kn}{N} \rceil}$
$S_{10} = X_{B1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{1\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{1\lceil \frac{Kn}{N} \rceil}$
$S_{20} = X_{B1}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{2\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{2\lceil \frac{Kn}{N} \rceil}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$S_{m0} = X_{B1}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 1}$	$S_{m\lceil \frac{(K-1)n}{N} \rceil + 2}$	$\cdots$	$S_{m\lceil \frac{Kn}{N} \rceil}$

Fig. 9. Simplex table decomposition

1. Choose the leading column from non-basic coefficients of the objective function (based on some common criteria each process selects from the columns it has).

2. The global exchange between the processes with the values obtained in step 1 and choosing optimal (pivot column for all processes).

3. Process that holds leading column chooses what variable to remove from basic ones (the choice of the leading line).

4. Process holding pivot column globally distribute numbers of the variables being included and excluded from the basic variables, as well as the pivot column.

5. Each process carries out the computation of a new canonical form by the rules of simplex method on the columns it has.

From the above we can conclude that the parallel version of algorithm is not much more complicated than sequential. It uses only minimum number of collective communications. All that should lead to a uniform loading of the system and high efficiency of parallelization [8].

The main difficulty arises when procedure of generating the basic plan is introduced. If simply add the necessary slack and dummy variables, and discard that columns after the first phase, the load will not be uniform. This problem can be solved in two ways:

- Redistribution of columns after the first stage, that is difficult and costly.

- Distribute the matrix in a way that each process got approximately equal number of columns of the original problem and columns that appear during the computation of the initial basic plan. This procedure requires for each process to know how many columns to discard after the first phase.

In the case of revised simplex method, the original matrix must be available to all processes because it is unknown what variables become basic and by which process they will be handled. Additional overhead for communication during computations lead to the fact that one cannot create an simple effective parallel version of the revised simplex method algorithm [9].

3.4 Algorithm of Parallel Simplex Method

This approach to parallelization of simplex method was implemented as MPI program Plinpex (parallel lineal exact solver). It uses rational data types from GNU MP library for exact computations or arbitrary precision floating point interval data types from MPFI library

(depends on compilation flags). A set of gcc 4.4.3 compiler, gdb debugger, efence and valgrind profiler was used. Implementation, testing, and some computational experiments were performed on gentoo gnu/linux based cluster of workstations that we built from computer class resources of department laboratory.

Algorithm Plinpex:

1: for each of N processes initialize MPI environment and identify itself via its MPI *rank*.
2: **if** $rank = 0$ **then**
3: read input problem file in MPS [10] format;
4: parse MPS file and save variable names;
5: initialize and fill matrix A, vector of objective function coefficients c and linear constraints b;
6: expand matrix A with slack and dummy variables for basic plan finding; initialize basic variables vector, dummy objective function;
7: **end if**
8: call MPI barier and initiate main solve function
9: broadcast common problem data from *rank* 0 process (problem; size and the number of slack and dummy variables);
10: evaluate number of main and dummy columns by knowing its *rank* and total number of processes N;
11: **if** $rank <> 0$ **then**
12: initialize memory for the part of A matrix, vectors of linear constraints b, dummy and objective function c, basic variables vector;
13: **end if**
14: broadcast the linear constraints and basic variables vector from *rank* 0 process;
15: **if** $rank = 0$ **then**
16: **for** $i = 1$ to N **do**
17: send part of main and dummy columns of A matrix to *rank* i;
18: **end for**
19: **else**
20: receive its main and dummy columns of A matrix;
21: **end if**
22: call MPI barrier to synchronize of main computation loop
23: **repeat**
24: choose the pivot column from columns this process handles;
25: call MPI all reduce and choose pivot column globally, let's assume the process that handles pivot column has *rank* L;
26: **if** minimum found **then**
27: **if** step one **then**
28: basic plan found, exclude dummy column and replace dummy objective function with primary one;
29: **else**
30: solution is found, goto 42;
31: **end if**
32: **end if**
33: **if** $rank = L$ **then**
34: chose a variable excluding from basic variables and pivot row;

35: **end if**
36: indexes of including and excluding to/from basic variables and broadcast the pivot column from *rank* L to other processes;
37: apply simplex method transformation on columns handled;
38: **if** entering/excluding basic variables are handled locally **then**
39: update basic variables vector;
40: **end if**
41: **until** solution is found
42: **if** $rank = 0$ **then**
43: output the problem solution;
44: **end if**
45: terminate MPI environment and exit.

4 Computational Experiments

Computational experiments were performed on "SKIF Ural" cluster of South-Ural State University. Brief specifications are presented in table 2.

Table 2

The specifications of computational platform

CPU type (per 1 blade)	2 quad core Intel Xeon E5472 3.0 GHz
System Memory (per 1 blade)	8 GB
Network type	InfiniBand (20Gbit/s, max latency of 2 ms)
Operating System	SUSE Linux Enterprise Server 10 x86_64

4.1 Experiment with Guaranteed Accuracy Floating Point Types

Parallel version of Gauss-Jordan elimination algorithm adapted for computing with mpfi_t (mpfin_t) and mpf_t (mpfn_t) was used. The effectiveness of parallelization is shown on fig. 10.

It should also be noted that when precision (mantissa length) is increased the efficiency of parallelization is also increased (table 3).

4.2 Linear Programming Problems Solving Experiment

Linear programming problems from Netlib library [11] were used as input data for computational experiment. This library contains complex problems that are often used for testing linear programming solving software systems. For the experiment we chose problems with various density and ratio.

Results with exact rational mpq_t type are presented on figure 11 , arbitrary precision floating point data types (mpf_t, mpfn_t) and interval data types (mpfi_t, mpfin_t) are shown on figure 12.

5 Conclusion

Methods for the effective application of arbitrary precision floating point (interval) data types in MPI environment in this paper are suggested in the work. Interval arbitrary precision types

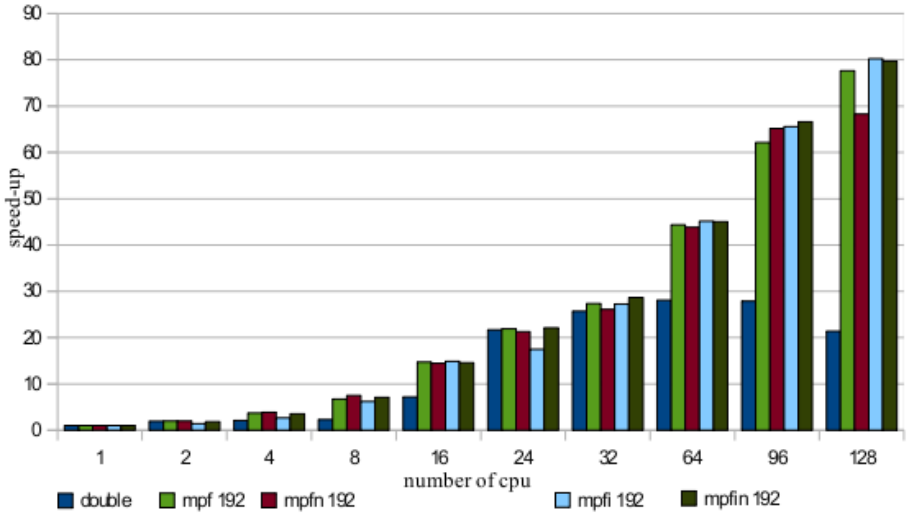

Fig. 10. Effectiveness of Guaranteed Accuracy Floating Point parallelization for Gauss-Jordan elimination

Table 3

Time resources for Gauss-Jordan elimination

	double	mpf			mpfn			mpfi	mpfin
N	**53**	**64**	**192**	**704**	**64**	**192**	**704**	**192**	**192**
1	43.04	1250.99	1817.42	6203.19	1238.50	1796.03	6219.70	3943.99	3817.11
2	22.06	627.28	913.78	3106.40	617.31	910.50	3132.24	2747.88	2073.00
4	20.41	315.35	489.06	1561.31	312.38	463.75	1573.92	1464.61	1072.29
8	18.83	171.44	269.83	795.95	166.03	239.94	804.95	634.02	537.36
16	5.98	86.91	123.35	413.09	85.19	124.01	410.43	265.07	261.93
24	1.98	60.12	83.07	325.72	61.50	84.44	277.53	225.70	172.57
32	1.68	47.70	66.41	212.97	47.30	68.72	220.01	144.66	133.10
64	1.53	27.90	40.98	135.55	28.06	40.97	123.79	87.36	84.79
96	1.54	19.85	29.25	92.61	19.88	27.57	84.55	60.14	57.31
128	2.01	16.86	23.41	77.10	18.53	26.31	76.18	49.15	47.90

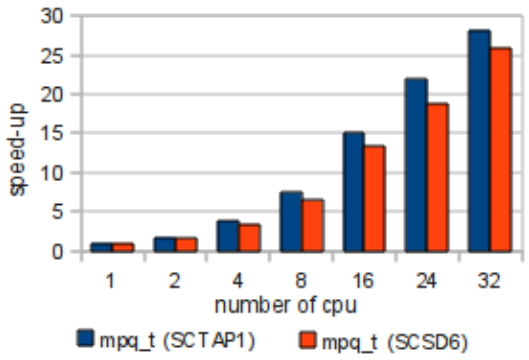

Fig. 11. effectiveness of rational type parallelization (lp)

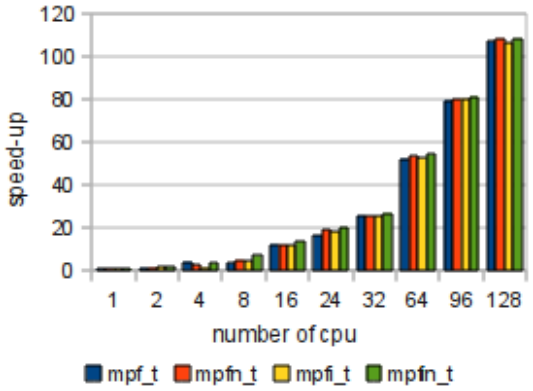

Fig. 12. Effectiveness of arbitrary precision floating point parallelization (LP)

give advantages of obtaining guaranteed results, that include rounding errors which can be analyzed to decide whether current precision of solution is sufficient. Interval arithmetic has main disadvantage of enclosures being too large but with arbitrary precision types we can cope with this problem to some extent and solve even high dimension problems. It should also be noted that the positive effect of parallelization in this case is not only in computation time reduction but also in ability to solve problems of higher dimension, because it is easy enough to reach memory limits, when the matrix will not fit entirely in memory of the single node.

All reviewed commercial programs use basic floating point data types and therefore they can not guarantee the accuracy of solutions. An exception from a number of programs, that does not provide guaranteed results are two open source implementations of simplex method, that use exact rational computations algorithms based on GNU MP library; and that fact inevitably leads to a substantial increase in computation time. However, they do not take advantage of parallel programming techniques which, with a skilful use, reduce the computation time and increase the number of problems that can be solved (problems with more dimensions).

The aim of this work was implementation of exact rational and guaranteed arbitrary precision floating point interval computations software for parallel and distributed computing systems called Plinpex. Software Plinpex use proposed methods for data types adaptation to MPI and parallel simplex method algorithm. The proposed algorithm uses only two collective communication at each iteration of the main computational loop of the program.

The computational experiments showed that the implemented methods are effective for problems of different dimensions, ratio and density. The usage of rational and arbitrary precision floating point interval data types adaptation to MPI gives ability to obtain exact and guaranteed results respectively. Examination of computational experiment result reveals efficiency of implemented parallel simplex method algorithm. According to experiments results the efficiency of parallelization depends on precision and it is about 70-80% (and grows with precision increase), and even higher for exact rational computations. However, the total time of computations can be improved with several algorithm optimizations. It is the subject for the further work.

References

1. MPI: A Message Passing Interface Standard, 1995. URL: http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html.

2. GNU Multiple Precision Arithmetic Library, 2010. URL: http://gmplib.org/.

3. MPFI library, 2008. URL: http://perso.ens-lyon.fr/nathalie.revol/software.html.

4. *Panyukov A.V., Germanenko M.I., Gorbik V.V.* Parallel algorithms for solving systems of linear algebraic equations using calculations without rounding//Parallel Computing Technologies (Pavt'2007). 2007. V. 2. P. 238-249.

5. *Panyukov A.V., Gorbik V.V.* Exact solution of linear programming problems on multiprocessor systems//Parallel Computing Technologies (Pavt'2008). 2008. P. 364-369.

6. MPFR library, 2009. URL: http://www.mpfr.org/.

7. *Pan V.Y., Reif J.H.* Fast and Efficient Parallel Linear Programming and Linear Least Squares Computations //Proceedings of the VLSI Algorithms and Architectures, Aegean Worksho on Computing. Springer-Verlag, 1986. P. 283-295.

8. *Hall Ju.* Towards a practical parallelisation of the simplex method // J. Computational Management Science. 2010. V. 7. N. 2. P. 139-170.

9. *Yarmish G.G.* A Distributed Implementation of the Simplex Method//UMI. Dissertations Publishing, 2001.

10. MPS format, 2008. URL: http://softlib.cs.rice.edu/pub/miplib/mps_format.

11. Netlib library collection, 1996. URL: ftp://netlib2.cs.utk.edu/lp/data.

GRATITUDES: The work is supported by Russian Foundation of Bounded Research (project 10-07-96003-r_ural_a).

Accepted for publication 7.06.2010.

РЕШЕНИЕ ЗАДАЧИ ЛИНЕЙНОГО ПРОГРАММИРОВАНИЯ С ПРОИЗВОЛЬНОЙ ТОЧНОСТЬЮ НА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ С MPI

© **Анатолий Васильевич Панюков**
Южно-Уральский государственный университет, пр. Ленина, 76, Челябинск, 454080, Россия, доктор физико-математических наук, профессор, зав. кафедрой экономико-математических методов и статистики, e-mail: a_panykov@mail.ru

© **Василий Владимирович Горбик**
Южно-Уральский государственный университет, пр. Ленина, 76, Челябинск, 454080, Россия, аспирант кафедры экономико-математических методов и статистики, e-mail: vgorbik@gmail.com

Ключевые слова: линейное программирование; метод симплекс-таблиц; распределенные вычисления; параллельная оптимизация; дробно-рациональные вычисления; произвольная точность; интервальная арифметика.

Предметом статьи являются способы получения как точного решения, так и приближенного решения с гарантированной точностью, а также способы повышения точности вычислений на распределенных вычислительных системах с MPI. Для получения таких решений применяются дробно-рациональные вычисления без округления, вычисления над числами с плавающей точкой произвольной наперед заданной точностью и интервальные вычисления с такими числами. Представлены способы адаптации предложенных типов данных к MPI. Результаты вычислительного эксперимента на разработанных параллельных алгоритмах решения систем линейных алгебраических уравнений и задач линейного программирования показывают эффективность их применения.

PARALLELIZED COMPUTATION OF EXTENDED UNIVERSAL GRÖBNER BASIS

© **Dmitry Alekseevich Pavlov**
Saint-Petersburg State Polytechnical University, Polytechnicheskaya 29, St.-Petersburg, 195251, Russia, Post-graduate Student of Applied Mathematics Department,
e-mail: dmitry.pavlov@gmail.com

Key words: universal Gröbner basis; polynomial ideal; Young diagram.

The article presents an algorithm to calculate Extended Universal Gröbner Basis (EUGB), working on wide range of polynomial ideals. The EUGB($\mathfrak{A}$) of a polynomial ideal $\mathfrak{A}$ is defined as a finite [1] set of polynomials $\{f_i\}$ whose Young diagrams $Y(f_i)$ meet the following condition: $\dim(\mathcal{L}(Y(f_i))\cap\mathfrak{A}) = 1$ (where $\mathcal{L}$ denotes the span of a set of polynomials in the quotient algebra of the ideal). It is known that the EUGB contains the Universal Gröbner Basis. The algorithm is based on geometry of Young diagrams in $\mathbb{Z}_+^d$, and finds the polynomials of EUGB mostly independently, which makes it able to run in parallel. An outline of the parallel version of the algorithm is given.

1 Notation and Background

Let $K[x_1,\dots,x_d]$ be a polynomial ring over a field K in d variables $X = \{x_1,\dots,x_d\}$. The space of monomials in these variables can be trivially identified with the lattice $\mathbb{Z}_{\geqslant 0}^d$. Here and after, we make no difference between the monomials and integer vectors with nonnegative coordinates—the elements of the lattice.

A polynomial ideal [2], generated by polynomials $(f_1,\dots,f_s)$, is defined as the following infinite set of polynomials from $K[x_1,\dots,x_d]$:

$$\langle f_1,\dots f_s\rangle = \left\{\sum_{i=1}^{s} h_i f_i : h_1,\dots,h_s \in K[x_1,\dots,x_d]\right\}.$$

It is known that the ideal can have more than one possible set of generators, each of which is called a basis of the ideal [2].

Let $\succ$ be a total order on $\mathbb{Z}_{\geqslant 0}^d$. It is called admissible, when it meets the following condition:

- $\alpha \succ 0$ for each $\alpha \neq \mathbf{0}$;
- If $\alpha \succ \beta$, then $\alpha+\gamma \succ \beta+\gamma$ for each $\gamma \in \mathbb{Z}_{\geqslant 0}^d$.

We denote as $\mathrm{LT}_\succ(f)$ the *leading term* of the polynomial f, that is, the term whose monomial is the biggest according to the admissible ordering $\succ$.

Let $\mathfrak{A}$ be a polynomial ideal in $K[x_1,\dots,x_d]$ and $\succ$ be an admissible monomial ordering. A finite set of polynomials $G \in \mathfrak{A}$ is called a Gröbner basis $\mathrm{GB}_\succ(\mathfrak{A})$ of $\mathfrak{A}$, if the leading term of any polynomial from $\mathfrak{A}$ is a multiple of some leading term of a basis polynomial [2]:

$$\langle\{\mathrm{LT}_\succ(g_i) : g_i \in G\}\rangle = \langle\{\mathrm{LT}_\succ(f) : f \in \mathfrak{A}\}\rangle.$$

The Gröbner basis G of the ideal $\mathfrak{A}$ has two important properties: first, it generates $\mathfrak{A}$, and second, every polynomial $g \in \mathfrak{A}$ has a *normal form* r, defined as a result of polynomial reduction of g w.r.t. G with the monomial order $\succ$:

$$g = h_1 f_1 + \cdots + h_s f_s + r, \qquad h_i, r \in K[x_1, \ldots, x_d].$$

By the definition of polynomial reduction, no term of r is a multiple of any of $\mathrm{LT}_\succ(f_i)$ (*).

2 Coideals, quotient algebra, and FGLM algorithm

The nondivisibility condition (*) has a convenient geometrical interpretation: all monomials of r are positioned "under" the monomial ideal, formed by $\{\mathrm{LT}(f_i), f_i \in \mathrm{GB}_\succ(\mathfrak{A})\}$. That is, they belong to the *coideal* $\mathrm{Co}(GB_\succ(\mathfrak{A})) = \mathbb{Z}^d_{\geqslant 0} \backslash \langle\{\mathrm{LT}(f_i)\}\rangle$.

All possible normal forms of polynomials of $\mathfrak{A}$ w.r.t. $\mathrm{GB}_\succ(\mathfrak{A})$ belong to $\mathcal{L}(\mathrm{Co}(\mathrm{GB}_\succ(A)))$, where $\mathcal{L}$ denotes the span of a set of polynomials in the quotient algebra of the ideal. We denote it $Q_\succ(\mathfrak{A})$, as it actually determines the quotient algebra of the ideal $\mathfrak{A}$ (fig. 2):

$$Q_\succ(\mathfrak{A}) \sim K[x_1, \ldots, x_d]/\mathfrak{A}.$$

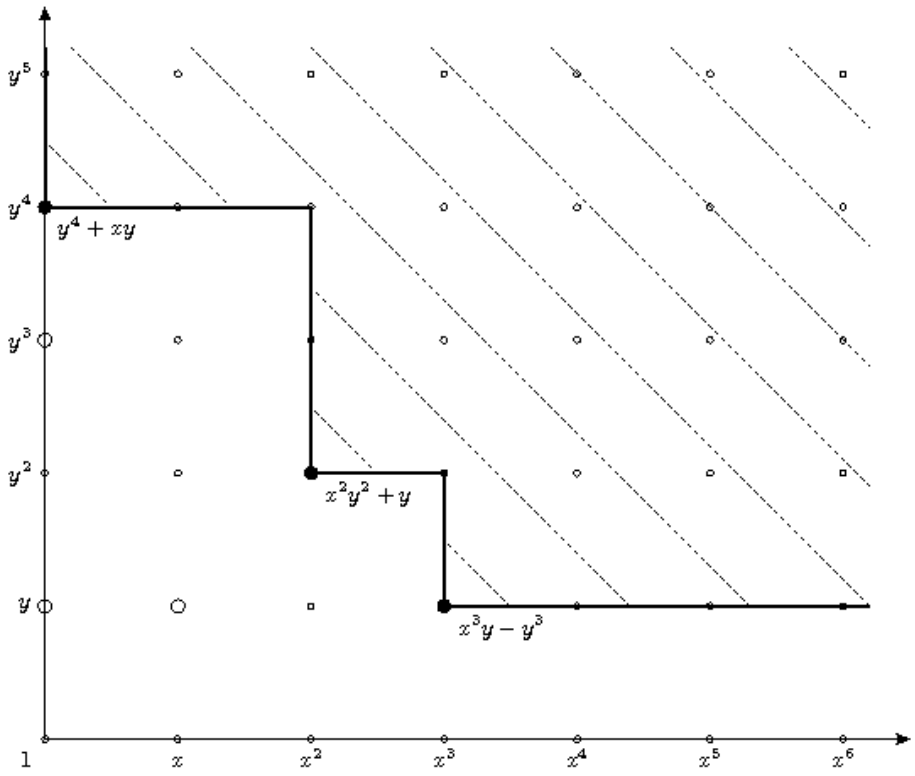

Fig. 1. The quotient algebra of an ideal generated by $\{y^4 + xy, x^2y^2 + y, x^3y - y^3\}$, is in turn generated by monomials, underlying the dashed area

We call monomials $\{m_i\}$ (not necessarily finite set) *linearly independent* w.r.t. $\mathfrak{A}$, if the intersection of their span with $\mathfrak{A}$ is also zero:

$$\mathcal{L}(\{m_i\} \cap \mathfrak{A}) = \{\mathbf{0}\}.$$

The ideal $\mathfrak{A}$ clearly do not contain any normal forms w.r.t. $\mathfrak{A}$, except the zero polynomial:

$$Q_{\succ}\mathfrak{A} \cap \mathfrak{A} = \{\mathbf{0}\}.$$

Hence, the monomials from the coideal $\mathrm{Co}(\mathrm{GB}_{\succ}(A))$ form a linearly independent set. But as soon as we add to this set the leading monomial of any polynomial of the Gröbner basis, we have this polynomial within the linear span of the set:

$$\mathcal{L}(Q_{\succ}(\mathfrak{A}) \cup \mathrm{LT}(f_i)) \cap \mathfrak{A} = \langle f_i \rangle,$$

and the dimension of this span is obviously equal to 1:

$$\dim(\mathcal{L}(Q_{\succ}(\mathfrak{A}) \cup \mathrm{LT}(f_i)) \cap \mathfrak{A}) = 1.$$

This way of Gröbner basis polynomials construction is used in FGLM [4] algorithm, which computes a Gröbner basis for an arbitrary monomial order $\succ'$, given another Gröbner basis for another monomial order $\succ$. Basically, the FGLM algorithms incrementally builds the coideal (starting from zero monomial), following the monomial order $\succ'$, until the monomials are not linearly independent. After they become linearly dependent, the corresponding polynomial of the Gröbner basis it calculated, and then the algorithms steps back and goes on, never adding that "linearly dependent" monomial again.

The FGLM algorithm has a limitation: it accepts only *zero-dimensional* polynomial ideals—the ideals whose quotient algebra is generated with a finite number of monomials. In another words, the coideal $\mathrm{Co}(\mathrm{GB}_{\succ}(A))$ in this case is finite, i.e. zero-dimensional.

3 Universal Gröbner basis and Young diagrams

We denote as $\mathrm{UGB}(\mathfrak{A})$ the *universal Gröbner basis* of the ideal $\mathfrak{A}$: the union of all possible Gröbner basiss with all admissible monomial orders. Robbiano [3] has shown that the UGB is always finite.

We define a d-dimensional Young diagram as a subset of $\mathbb{Z}^d_{\geqslant 0}$ lattice, with the only requirement that if it contains some monomial m, it must also contain all divisors of m.

We call a Young diagram of a polynomial $r \in \mathfrak{A}$ the one formed by the terms (monomials) of r, that is, a union of monomials of r and all their divisors (fig. 3).

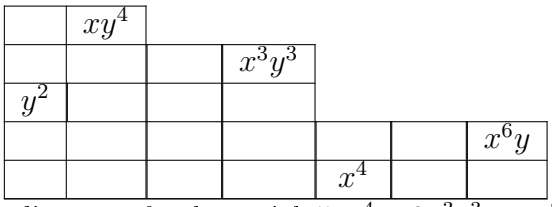

Fig. 2. Young diagram of polynomial $5xy^4 + 2x^3y^3 + xy^2 - x^6y + 8x^4$

For every polynomial $f \in \mathrm{UGB}(\mathfrak{A})$ the following condition is hold [1]:

$$\dim(\mathcal{L}(Y(f) \cap \mathfrak{A})) = 1. \tag{1}$$

This condition does not guarantee that the polynomial f is a part of $\mathrm{UGB}(\mathfrak{A})$; nevertheless, this condition is easier to check, as it does not imply any admissible monomial order.

We denote $\mathrm{EUGB}(\mathfrak{A})$ the *Extended Universal Gröbner basis* of the ideal $\mathfrak{A}$:

$$\mathrm{EUGB}(\mathfrak{A}) = \{f \in \mathfrak{A} : \dim(\mathcal{L}(Y(f) \cap \mathfrak{A})) = 1\}.$$

The $\mathrm{EUGB}(\mathfrak{A})$ is always finite [1]. Clearly, $\mathrm{UGB}(\mathfrak{A}) \subset \mathrm{EUGB}(\mathfrak{A})$. Unlike the finding of $\mathrm{UGB}(\mathfrak{A})$, the finding of $\mathrm{EUGB}(\mathfrak{A})$ is done via geometrical and combinatorial operation on $\mathbb{Z}^d_{\geqslant 0}$.

4 Finding EUGB($\mathfrak{A}$)

The described algorithm searches for Young diagrams whose spans have a one-dimensional intersection with the ideal $\mathfrak{A}$, and this intersection is itself an ideal generated by some polynomial from $\mathrm{EUGB}(\mathfrak{A})$.

The following global variables are used:

- `*basis*` — an arbitrary Gröbner basis to start with; for example, a Gröbner basis w.r.t. **degrevlex** monomial order. It is needed for checking the linear independence of the monomials of Young diagrams.
- `*diagrams*` — a list of found diagrams that fulfill the condition (1). Each diagram is defined by a list of nondivisor monomials (i.e. "corners" of the diagram. At the start, the list of diagrams contain the Young diagrams of the polynomials of the `*basis*`. (Although it could be empty, but then the procedure would have taken more time.)
- `*bad-coideals*` — a list of coideals that do not contain diagrams of interest of size less than `MAX-SIZE`. At the start, this list is empty.

The following helper functions are mentioned but not listed:

- `nondivisors (poly)`: accepts a polynomial and returns its monomials, that are not divisors of any other monomials of this polynomial.
- `contains-divisors-of (monomials, diagram)` finds among the `monomials` the divisors of `diagram`'s"corners".
- `remove-multiples-of (monomials, corner)` removes from `monomials` the ones that are multiples of `corner`.
- `intersect-with-ideal (sequence, basis)` checks the intersection of a span of the `sequence` and the ideal, generated by `basis`. If the intersection is not zero, it is assumed 1-dimensional, and the resulting generating polynomial is returned.
- `coideal-belongs (inner, outer)` checks that the `inner` monomial coideal is a subset of the `outer`.

- `closest-to-origin (list)` returns the monomial from the `list` that is closest to the origin.

As the first step, the algorithm outputs the polynomials of the given `*basis*`, which clearly meet the condition (1), and saves the diagrams of these polynomials. After that, the monomial coideals not containing the found `*diagrams*` are enumerated.

```
find-eugb (*basis*):
  *diagrams* ← {}
  for all poly ∈ *basis* do
    yield poly
    *diagrams* ← *diagrams* ∪ nondivisors(poly)
  end for
  repeat
    oldsize = size(*diagrams*)
    process-coideals(*diagrams*)
  until size(*diagrams*) = oldsize
```

In order to prevent duplicating diagrams in the output, each diagram is being searched for in a monomial coideal, which does *not* contain at least one "corner" of already found `*diagrams*`. Such a coideal (there can be many of them, but not infinitely many) is computed by the recursive procedure `process-coideals`. Once it is found, the `find-polynomial` function is invoked for this coideal.

```
process-coideals (diagrams, coideal = {}):
  if diagrams = {} then
    find-polynomial(coideal)
  else
    diagram ← any of the diagrams
    if contains-divisors-of(coideal, diagram) then
      process-coideals( diagrams\diagram , coideal)
    else
      for all corner ∈ diagram do
        new-coideal ← corner ∪ remove-multiples-of(coideal, corner)
        process-coideals( diagrams\diagram , new-coideal)
      end for
    end if
  end if
```

The next procedure accepts a coideal and grows a Young diagram inside it, starting from an empty diagram, and adding monomials one-by-one, until the condition 1 is met. On each step, from all monomials ("dimples") available for addition, the one closest to the origin is selected.

As the coideal may be infinite (especially in case we are dealing with a non-zero-dimension polynomial ideal), the procedure is forced to stop once the size of the coideal reaches `MAX-SIZE`.

```
find-polynomial (coideal):
  for all bad-coideal ∈ *bad-coideals* do
    if coideal-belongs(coideal, bad-coideal) then
      return
```

```
    end if
  end for
  seq ← {}
  dimples ← {0}
  while dimples ≠ {} do
    if |seq| > MAX-SIZE then
      print Linear-dependent Young diagram not found in coideal.
      *bad-coideals* = *bad-coideals* ∪ coideal
      return
    end if
    new-monom ← closest-to-origin(dimples)
    seq ← seq ∪ new-monom
    poly ← intersect-with-ideal(seq, *basis*)
    if poly ≠ 1 then
      *diagrams* ← *diagrams* ∪ support(poly)
      yield poly
    end if
    dimples ← update-dimples(new-monom, dimples\new-monom)
  end while
```

The helper procedure `update-dimples` adds a new "dimple" to the list of available monomials for the next step of diagram growing, and removes its divisors from the list.

```
update-dimples (dimples new-cell):
  for all v ∈ X do
    if ∄c ∈ dimples : c ≺ v · new-cell then
      dimples ← dimples ∪ v · new-cell
    end if
  end for
  return dimples
```

The above algorithm is able to find the polynomials of EUGB($\mathfrak{A}$), whose Young diagrams are of size less than `MAX-SIZE`. The size limit can be discarded for zero-dimensional ideals, where the endless growing of a diagram of linearly independent monomials is theoretically impossible. On all other ideals, the size limit allows to avoid endless loops, but can lead to loss of some EUGB polynomials with too big Young diagrams.

Unfortunately, there is no possibility to give any reasonable estimation for `MAX-SIZE` that would guarantee the generation of entire EUGB. Obviously, the size of a Young diagram of a polynomial can not be less than its degree; and the best known estimations on the degree of Gröbner basis elements are 2^{2^k} for **lex** ordering [5] and k^2+1 for **grevlex** ordering [6] (where k is the maximum degree amongst the generators of the ideal).

5 Parallelizing the EUGB finding algorithm

A lot of operations in the algorithm described above can be run in parallel. While `process-coideals` in the original algorithms outline is called step-by-step, with each next step having one more diagram, it is possible first to calculate a few coideals to search in, then search for EUGB polynomials in each coideal independently, using a separate working thread. The following statements should be considered when implementing the parallelized EUGB finding:

- It is possible that the parallel algorithm would find the same Young diagrams simultaneously in different computing threads. While this overhead is not likely to be completely eliminated, some techniques would help (see below).
- Each coideal given to the working thread should not contain any of the `*diagrams*` found so far. Also, the less monomials the "current" processed coideals have in common, the better.
- To further decrease the possibility of finding duplicate diagrams simultaneously, the following principle should be used when selecting the next monomial for diagram increment (in addition to `closest-to-origin`): once the monomial got into the diagram being built by a working thread, it is better not to add this monomial to a diagram being build in a neighbor working thread; this should be done only if no other options are left.
- Once all the working threads are done with their Young diagrams and return some polynomials, the control flow should go back to the main thread, where the results are stored, the duplicates are eliminated, and the new portion of monomial coideals is calculated for the next parallel computation step.

6 Acknowledgements

I wish to thank Nickolay Vasiliev for encouraging and supervising my work on this topic.

References

1. *Vasiliev N.* Monomial Orderings, Young Diagrams and Gröebner Bases // Proceedings of the International Conference "Computer Algebra in Scientific Computing" (CASC). Technical University of Munchen. Munchen, 2003.
2. *Cox D.A., Little J.B., O'Shea D.* Ideals, Varieties, and Algorithms // Introduction to Computational Algebraic Geometry and Commutative Algebra. New York: Springer, 2007.
3. *Robbiano L.* Term ordering on the polynomial ring // Proceedings of EUROCAL '85 (Linz), Lecture Notes in Computer Science. 1985. V. 204. P. 513-517.
4. *Faugère J. C., Gianni P., Lazard D., Mora T.* Efficient computation of zero-dimensional Gröbner bases by change of ordering // J. of Symbolic Computation. 1993. V. 16. Issue 4. P. 329-344.
5. *Mayr E., Meyer A.* The complexity of the word problem for commutative semigroups and polynomial ideals // Adv. Math. 1982. V. 46. P. 305-329.
6. *Lazard D.* Gröbner Bases, Gaussian elimination and resolution of systems of algebraic equations // Proceedings of the European Computer Algebra Conference on Computer Algebra (EUROCAL). London: Springer-Verlag, 1983. P. 146-156.

Accepted for publication 7.06.2010.

ПАРАЛЛЕЛЬНОЕ ВЫЧИСЛЕНИЕ РАСШИРЕННЫХ УНИВЕРСАЛЬНЫХ БАЗИСОВ ГРЁБНЕРА

© **Дмитрий Алексеевич Павлов**
Санкт-Петербургский государственный политехнический университет, Политехническая, 29, Санкт-Петербург, 195251, Россия, аспирант кафедры прикладной математики,
e-mail: dmitry.pavlov@gmail.com

Ключевые слова: универсальный базис Грёбнера; полиномиальный идеал; диаграмма Юнга.

В статье представлен алгоритм вычисления расширенного универсального базиса Грёбнера (EUGB), работающий на широком классе полиномиальных идеалов. EUGB($\mathfrak{A}$) полиномиального идеала $\mathfrak{A}$ определён как конечное [1] множество полиномов f_i, чьи диаграммы Юнга $Y(f_i)$ удовлетворяют следующему условию: $\dim(\mathcal{L}(Y(f_i)) \cap \mathfrak{A}) = 1$ (где $\mathcal{L}$ обозначает линейную оболочку множества полиномов в факторалгебре идеала). Известно, что EUGB содержит универсальный базис Грёбнера идеала. Алгоритм основан на геометрических свойствах диаграмм Юнга в $\mathbb{Z}_+^d$, и элементы EUGB находятся им по большей части независимо друг от друга, что позволяет вычислять их параллельно. В последней части статьи дана схема параллелизации алгоритма.

UDK 519.688

PARALLEL ALGORITHMS FOR COMPUTING THE CHARACTERISTIC POLYNOMIALS BASED ON THE METHOD OF HOMOMORPHIC IMAGES

© **Oksana Nikolayevna Pereslavtseva**
Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000, Russia, programmer of Algebraic Computing Department,
e-mail: Pereclavtseva@rambler.ru

Key words: computing characteristic polynomial of matrices; parallel algorithm; method of homomorphic images; cluster.

There are produced parallel algorithms for computing the characteristic polynomials for integer and polynomial dense matrices. The algorithms are based on the method of homomorphic images in the ring of integers and in the ring of polynomials.We have obtained an upper bound for numerical coefficients of a characteristic polynomial. There are stated and discussed results of experiments with parallel algorithms for computing the characteristic polynomials of integer and polynomials matrices. The experiments with parallel algorithm are conducted on cluster MVS100k of Joint Super-Computer Center RAS.

1 Introduction

Computation of the characteristic polynomials for dense matrices is a classical problem of computing algebra. Let's give overview of the basic results.

In 1881 Leverrie suggested one of the first methods for computing the characteristic polynomials of matrices over ring[1]. Faddeev D.K. in 1943 has offered modification of Leverrie's method [2]. This method also can compute an adjoint matrix. The Leverrie's algorithm (with Winograd's improvement [3] (p.656)) demands $\sim 4n^{3.5}$ ring operations, Faddeev's algorithm demands $\sim 2n^4$ ring operations for computing the characteristic polynomial of the matrix of order $n \times n$. The basis of these algorithms is computation of matrix degrees. It allows to use parallel matrix multiplication to obtain the parallel algorithms for computing the characteristic polynomials. We notice that till now Leverrie's and Faddeev's algorithms have been the best parallel algorithms although much improvement of consecutive algorithms for computation of characteristic polynomials have been done.

Seifullin's algorithm (2002) [4] has less ring operations ($\sim 1/2n^4$). But his algorithm cannot work parallel because it is strictly consecutive and is not recursive. For the same reason Malaschonok's algorithm (1999) [5] with complexity $\sim 8/3n^3$ and its modification (2008) [6] with complexity $\sim 7/3n^3$ also cannot be write in parallel form. These two algorithms have the least number of ring operations.

Danilewsky's algorithm (1937) [7], Keller-Gehrig's algorithm (1985) [8], Pernet-Storjohann's algorithm (2007) [9] are the best for computation of characteristic polynomials over a finite field. It demands $\sim 2n^3$, $O(n^\omega \log_2 n)$ and $O(n^\omega)$ operations over a finite field accordingly. Here $O(n^\omega)$ is a complexity of matrix multiplication. These algorithms are the asymptotic best algorithms for computation of characteristic polynomials over a ring of integers and over a ring of polynomials with integer coefficients if the CRT algorithm is used.

This work is directed to development of parallel methods for computation of characteristic polynomials of dense matrices. The considered parallel algorithms are based on the method of homomorphic images. It is do due to the fact that modular arithmetics assumes natural parallelism since computation of characteristic polynomials for each module is independent and parallel. In Section 2 there is a detailed description application of the method of homomorphic images to a ring of polynomials of many variables and to a ring of integers is described detail. In Section 3 an upper bound of numerical coefficients of a characteristic polynomial is obtained. This upper bound is necessary for application of the method of homomorphic images.

Algorithms for computation of a characteristic polynomial over a finite field for the different sizes of matrices will show different time. Therefore, it is needed to realize various algorithms for computation of characteristic polynomials, to compare them experimentally and to reveal the most effective ones. Some of the methods have been realized and the experiments have been made. In Section 4 the parallel algorithm for computation of characteristic polynomials of integer and polynomial matrices is described. In Section 5 results of the experiments with the parallel algorithms are discussed.

2 Application of the method of homomorphic images for characteristic polynomials computation

The method of homomorphic images is described in work [10]. We will apply to the method of homomorphic images for computation of characteristic polynomials of polynomial matrices of many variables.

The general circuit of a method of the homomorphic images applied to a ring of polynomials of many variables $\mathbb{Z}[x_1, \ldots, x_t]$ is the following.

Let $A = (a_{\mu\nu}(x_1, \ldots, x_t)),\ 1 \leqslant \mu \leqslant n,\ 1 \leqslant \nu \leqslant n,$ be a polynomial matrix, $A \in Z^{n\times n}[x_1, \ldots, x_t, y]$, $f = (-1)^n(y^n + \sum_{i=1}^{n} f_i(x_1, \ldots, x_t)y^{n-i})$ is its characteristic polynomial, $f \in Z[x_1, \ldots, x_t, y]$.

Let m_s be a hight degree of a variable x_s, $1 \leqslant s \leqslant t$ in polynomials f_i, $1 \leqslant i \leqslant n$ and β be a greatest absolute value of numerical coefficients.

0) Let's choose h prime numbers: $p_1, \ldots, p_h$ so that the inequality was fulfilled $\log_2 \beta < \log_2(p_1 \cdots p_h)$. Then we will pass to homomorphic images of elements $a_{\mu,\nu}$ at mappings

$$\mathbb{Z}[x_1, \ldots, x_t] \to \mathbb{Z}[x_1, \ldots, x_t]/p_i\mathbb{Z}[x_1, \ldots, x_t].$$

Denote

$$\mathbb{Z}[x_1, \ldots, x_t]/p_i\mathbb{Z}[x_1, \ldots, x_t] = \mathbb{Z}_{p_i}[x_1, \ldots, x_t].$$

1) Let's choose m_t polynomials: $x_t, x_t - 1, \ldots, x_t - (m_t - 1)$. Then we will pass to homomorphic images of elements $a_{\mu,\nu}$ at mappings

$$\mathbb{Z}_{p_i}[x_1, \ldots, x_t] \to \mathbb{Z}_{p_i}[x_1, \ldots, x_t]/(x_t - j)\mathbb{Z}_{p_i}[x_1, \ldots, x_t],$$

$(0 \leqslant j \leqslant m_t - 1)$. The following isomorphism takes place

$$\mathbb{Z}_{p_i}[x_1, \ldots, x_t]/(x_t - j)\mathbb{Z}_{p_i}[x_1, \ldots, x_t] \sim \mathbb{Z}_{p_i}[x_1, \ldots, x_{t-1}],$$

2) Let's choose m_{t-1} polynomials: $x_{t-1}, x_{t-1}-1, \ldots, x_{t-1}-(m_{t-1}-1)$. Then we will pass to homomorphic images of elements $a_{\mu,\nu}$ at mappings

$$\mathbb{Z}_{p_i}[x_1, \ldots, x_{t-1}] \to \mathbb{Z}_{p_i}[x_1, \ldots, x_{t-1}]/(x_{t-1}-j) \sim \mathbb{Z}_{p_i}[x_1, \ldots, x_{t-2}],$$

$(0 \leqslant j \leqslant m_{t-1}-1)$.

Let's continue toconstruct similarly the homomorphic images of elements $a_{\mu,\nu}$ for each variable $x_s, s = t-2., 1$. As a result we will pass to homomorphic images in $\mathbb{Z}_{p_i}$ and we will obtain $hm_1m_2\cdots m_t$ matrices

$$M_{ij_1\ldots j_t} \in \mathbb{Z}_{p_i}^{n\times n},\ 1 \leqslant i \leqslant h,\ 1 \leqslant j_1 \leqslant m_1,\ \ldots,\ 1 \leqslant j_t \leqslant m_t.$$

Let's calculate characteristic polynomials of matrices $M_{ij_1\ldots j_t}$ by means of some algorithm over a finite field. We will obtain $hm_1m_2\cdots m_t$ polynomials $f_{ij_1\ldots j_t}(y),\ 1 \leqslant i \leqslant h,\ 1 \leqslant j_1 \leqslant m_1,\ \ldots,\ 1 \leqslant j_t \leqslant m_t$.

Computation of a required characteristic polynomial is found by means of the Chinese remainder theorem upside-down.

Starting with m_t images

$$\{f_{ij_1\ldots j_{t-1}1}(y), f_{ij_1\ldots j_{t-1}2}(y), \ldots, f_{ij_1\ldots j_{t-1}m_t}(y)\}$$

of the polynomial $f_{ij_1\ldots j_{t-1}}(x_t, y)$ we will restore this polynomial by means of the Chinese remainder theorem.

Also starting with m_{t-1} images

$$\{f_{ij_1\ldots j_{t-2}1}(x_t, y), f_{ij_1\ldots j_{t-2}2}(x_t, y), \ldots, f_{ij_1\ldots j_{t-2}m_{t-1}}(x_t, y)\}$$

of the polynomial $f_{ij_1\ldots j_{t-2}}(x_{t-1}, x_t, y)$ we will restore this polynomial. And so on.

Having fulfilled restoring on all variables, we will obtain k polynomials

$$\{F_1(x_1, \ldots, x_t, y), \ldots, F_h(x_1, \ldots, x_t, y)\},$$

in factor rings $\mathbb{Z}_{p_1}, \ldots, \mathbb{Z}_{p_h}$ accordingly.

The characteristic polynomial of the matrix A is recovered on these h polynomials.

3 Upper bound of coefficients of characteristic polynomials over rings $\mathbb{Z}$ and $\mathbb{Z}[x_1, \ldots, x_t]$

For program realization of modular algorithm for computing of a characteristic polynomial of a matrix is necessary to know numerical modules $p_1, \ldots, p_h$ and polynomial modules $x_1, x_1-1, \ldots, x_1-(m_1-1); \ldots; x_t, x_t-1, \ldots, x_t-(m_t-1)$ for a polynomial matrix .

The best upper bound which is known today for the coefficients of a characteristic polynomial of an integer matrix is obtained in the work [11]. According to that the number of bits in the coefficients of a characteristic polynomial does not exceed

$$\mu_{n,a} = \frac{n}{2}(\log_2 n + 2\log_2 a + 0,22),$$

when n is order of the matrix, a is the greatest absolute value for numerical coefficients of matrix elements.

The array of prime numbers is supposed to be set. Choosing from that the prime numbers and calculating their product it is easy to choose sufficient number h of modules. The inequality

$$\mu_{n,a} \leqslant \log_2(p_1 p_2 \cdots p_h) \tag{1}$$

must be fulfilled.

Let's find upper bound for coefficients of a characteristic polynomial of a polynomial matrix from one variable [12].

Let $F(x,y) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m-1} g_{ij} x^j \right) y^i$. Numerical modules $p_1, \ldots, p_h$ should be selected so that $p_1 \cdots p_h > \max |g_{ij}|$.

Let $||f||$ be a norm of a polynomial f. It is the greatest absolute value for numerical coefficients of the polynomial f.

Let $A = (a_{ij}(x)),\ 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant n$,
$d = max\{\deg a_{ij}(x)\}$,
$||A|| = max\{||a_{ij}||\} = a$ for $1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant n$;
$s(A) = max\{s(a_{ij}(x))\} = t$.

Theorem 1 *Let*

$$F(x,y) = y^n + f_1(x)y^{n-1} + \cdots + f_n(x)$$

be the characteristic polynomial of matrix $A(x)$ *and* $m = \max\{\deg f_1(x), \ldots, \deg f_n(x)\} + 1$.

Then $m \leqslant nd+1$ *and for the greatest on the module of numerical coefficient of a polynomial* $F(x,y)$ *the inequality is carried out*

$$\log_2 ||F(x,y)|| \leqslant n(\log_2 n + \log_2 a + \log_2 t) - \log_2 t. \tag{2}$$

Proof 1 *The polynomial* $f_i(x)$ *is a sum of all the* $(n-i) \times (n-i)$ *diagonal minors of* $A(x)$. *Therefore* $m \leqslant nd + 1$.

In order to find an upper bound for $||F(x,y)||$ *we use Leverrie-Faddeev's algorithm [2]:*
$B_0 = E$;
$i = 1, \ldots, n$:
$\{A_i = AB_{i-1}$;
$f_i = (1/i)TrA_i$;
$B_i = A_i - f_i E\}$.

For matrix B_i *we consider two norms* $||B_i||_d$ *and* $||B_i||_n$. $||B_i||_d$ *is the greatest absolute value for numerical coefficients of matrix* B_i *which elements are on the main diagonal.* $||B_i||_n$ *is the greatest absolute value for numerical coefficients of matrix* B_i *which elements are not on the main diagonal.*

For $i = 1$: $||A_1|| \leqslant a,\ ||f_1|| \leqslant na,\ ||B_1||_n \leqslant a,\ ||B_1||_d \leqslant (n+1)a$.

For $i > 1$: $\min\{s(B_{i-1}), t\} = t$. *Then*
$||f_i|| \leqslant (n/i)||A_i||$;
$||A_i|| \leqslant (n-1)ta||B_{i-1}||_n + ta||B_{i-1}||_d$;
$||B_{i-1}||_n = ||A_{i-1}||$ *and* $||B_{i-1}||_d = ||A_{i-1}|| + ||f_{i-1}||$.
Then, $||A_i|| \leqslant ta(n||A_{i-1}|| + ||f_{i-1}||) \leqslant tan(i/(i-1))||A_{i-1}||$. *Then* $||A_i|| \leqslant n^{i-1}t^{i-1}a^i i$ *and* $||f_i|| \leqslant n^i t^{i-1} a^i$ *for* $i > 1$.

$||f_i||$ is greatest when $i = n$. $||f_i|| \leqslant n^n t^{n-1} a^n$. Taking the logarithm of the last inequality by the basis of 2, we btain the upper bound for numerical coefficients of characteristic polynomials of polynomial matrices of one variable (2).

Remark 2 *If polynomial f of one variable is dense then $s(f) = d + 1$.*

Remark 3 *The formula (2) is true for a dense polynomial matrix of many variables. The number of polynomial modules is calculated for each variable $x_1, \ldots, x_t$: $m_i = nd_i + 1$, where d_i is the hight degree of the variable x_i, $i = 1, \ldots, t$.*

4 Parallel algorithms for computing the characteristic polynomials which are based on the method of homomorphic images

4.1 The circuit of data communication

The matrix $A \in Z^{n \times n}[x_1, \ldots, x_t]$ and the number *boundlev* are input data for the parallel algorithm. The parameter *boundlev* is number of levels of the algorithm tree. This number depends on the task (the order and matrix coefficients) and on the computing cluster. The set of prime numbers is defined in advance and stored on each processor. At first the quantity h of numerical modules and the quantity of polynomial modules on each variable $x_1, \ldots, x_t$ is calculated.

A graph of the algorithm is a binary tree which is presented in a figure 1. Horizontal lines are divide the graph into levels. At the first level there is only a root, at the second level there are its two daughter nodes, at the third level there are their daughter nodes etc.

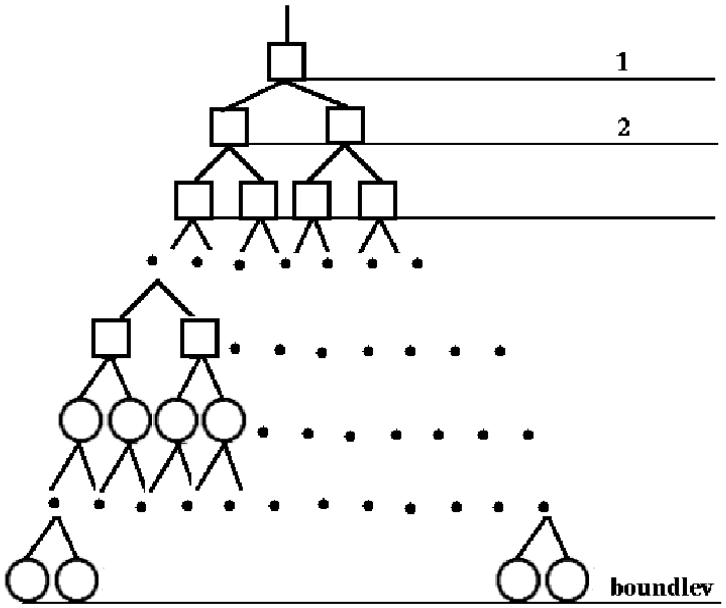

Fig. 1. The graph of the algorithm

In the input root receives the matrix A and the array

$$intervals = \{[1, m_1], \ldots, [1, m_t], [1, k]\}.$$

In the array *intervals* first two numbers $[1, m_1]$ correspond to polynomial modules $x_1, x_1 - 1, x_1 - 2, .., x_1 - (m_1 - 1)$, second two numbers $[1, m_2]$ correspond to polynomial modules $x_2, x_2 - 1, x_2 - 2, .., x_2 - (m_2 - 1)$ and so on, and the last two numbers $[1, h]$ correspond to numerical modules $[p_1, p_2, .., p_h]$. As a result of calculations in the root we will receive a characteristic polynomial of the matrix A.

Daughter nodes at level 2 receive from root a matrix A and the array of modules $intervals = \{[1, m_1], \ldots, [1, m_t], [1, h_1]\}$ (for the left node) or $intervals = \{[1, m_1], \ldots, [1, m_t]$, $[h_1 + 1, h\}$ (for the right node), where $h_1 = \lfloor (1 + h)/2 \rfloor$. Thus each daughter node has half of all numerical modules. It should return in root the characteristic polynomial of matrix A in a factor-ring module of products of all modules received from the root. For the left node the module is $p_1 p_2 \cdots p_{h_1}$, for the right node – $p_{h_1+1} p_{h_1+2} \cdots p_h$.

Each node at level 2 also divide the array of modules half-and-half and sends to their daughter nodes to level 3. This process proceeds, while there are free processors and on each processor is available more than one module.

The graph of the algorithm has 2 types of nodes. Nodes of 1st type correspond to numerical modules and are designated in figures by squares. Nodes of 2nd type correspond to polynomial modules and are designated in figures by circles.

Node of 1st type.

The node of 1st type with daughter nodes is shown in a figure 2.

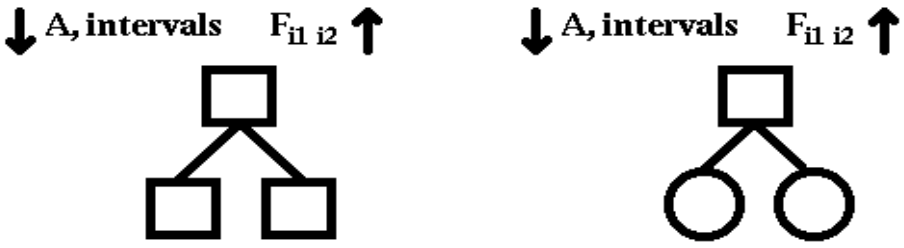

Fig. 2. Node of 1st type

The node of 1st type on an input receives a matrix A and the array $intervals = \{[1, m_1], \ldots, [1, m_t], [i_1, i_2]\}$. Numbers i_1 and i_2 set the first and last prime numbers from the list $\{p_1, \ldots, p_h\}$.

The node of 1st type builds a polynomial $F_{i_1 i_2}(x_1, \ldots, x_t, y)$ on the remainders received from daughter nodes. As a result it returns the polynomial $F_{i_1 i_2}$ which is the characteristic polynomial module $\{p_1, \ldots, p_h\}$.

The node of 1st type sends numerical modules to two daughter nodes. The left node on an input receives a matrix A and the array $intervals1 = \{[1, m_1], \ldots, [1, m_t], [i_1, i_s]\}$, the right node – the matrix A and the array $intervals2 = \{[1, m_1], \ldots, [1, m_t], [i_s + 1, i_2]\}$, where $i_s = \lfloor (i_1 + i_2)/2 \rfloor$. If the daughter node has received only one numerical module it is node of 2nd type, else it is node of 1st type.

The node of 1st type receives two polynomials $F_{i_1, i_s}(x_1, \ldots, x_t, y) \in \mathbb{Z}_{d1}[x_1, \ldots, x_t, y]$ and $F_{i_{s+1}, i_2}(x_1, \ldots, x_t, y) \in \mathbb{Z}_{d2}[x_1, \ldots, x_t, y]$ from daughter nodes, where $d1 = p_{i_1} \cdots p_{i_s}$ and $d2 = p_{i_{s+1}} \cdots p_{i_2}$. After reception of these polynomials the father node computes the polynomial $F_{i_1 i_2}(x_1, \ldots, x_t, y) \in \mathbb{Z}_{p_{i_1} \cdots p_{i_2}}[x_1, \ldots, x_t, y]$.

Node of 2nd type.

The node of 2nd type with daughter nodes is shown in a figure 3.

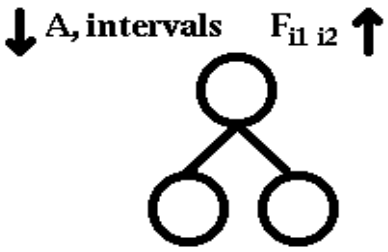

Fig. 3. Node of 2nd type

The node of 2nd type on an input receives a matrix A, the array

$$intervals = \{[1, m_1], \ldots, [b_s, e_s], [j_{s+1}, j_{s+1}], \ldots, [i, i]\}$$

and number s. Numbers i_1 and i_2 set the first and last prime numbers from the list $\{p_1, \ldots, p_h\}$. s denotes number of an active value, i gives number of a prime number from the set of modules $\{p_1, \ldots, p_h\}$, the interval $[b_s, e_s]$ gives polynomial modules of the active value x_s. If $v < s$ then $[b_v, e_v] = [1, m_v]$. If $v > s$ then $[b_v, e_v] = [j_v, j_v]$, i.e. the interval $[b_v, e_v]$ contains one module $x_v - (j_v + 1)$.

The node of 2nd type builds a polynomial $F_{b_s e_s} \in \mathbb{Z}_{p_i}[x_1, \ldots, x_s, y]$ on the polynomial remainders received from daughter nodes.

Daughter nodes for a node of 2nd type are nodes of 2nd type. The left node on an input receives a matrix A, the array $intervals1 = \{[1, m_1], \ldots, [b_s, h_s], [j_{s+1}, j_{s+1}], \ldots, [i, i]\}$ and the number $r1$ of an active value, where $h_s = (b_s + e_s)/2$. If $b_s < h_s$ then the number $r1 = s$ else $r1 = s - 1$. The right node – the matrix A, the array $intervals2 = \{[1, m_1], \ldots, [h_s + 1, e_s], [j_{s+1}, j_{s+1}], \ldots, [i, i]\}$ and the number $r2$. If $h_s + 1 < e_s$ then the number $r2 = s$ else $r2 = s - 1$.

The node of 2nd type receives two polynomials

$$f_1 \in \mathbb{Z}_{p_i}[x_1, \ldots, x_{r1}, y] \text{ and } f_2 \in \mathbb{Z}_{p_i}[x_1, \ldots, x_{r2}, y]$$

from daughter nodes. After reception of these polynomials the father node computes the polynomial $(f_i) \in \mathbb{Z}_{p_i}[x_1, \ldots, x_s, y]$.

4.2 Parallel algorithm

Let $A \in Z^{n \times n}[x_1, \ldots, x_t]$; k be the number of processors;
the **function** $numbOfMod()$ compute the number of polynomial and numerical modules. The **function** $numbOfMod()$ uses formulas (1), (2);
the **function** $send(a, b, ..., c, i)$ send a data $a, b, ..., c$ from the current processor to the processor i;
the **function** $recv(a, b, ..., c, i)$ receive a data $a, b, ..., c$ on the current processor from the processor i;
the **function** $go\_down(intervals, r, boundlev)$ divide the task into two parts;
the **function** $charPol(A, intervals, r)$ compute on one processor a polynomial g on one processor g fulfills if a characteristic polynomial of a matrix A to take product of module from $intervals$;
the **function** $recoveryNewton(f_1, f_2, r1, r2)$ compute by means the Chinese remainder theorem a polynomial f using remainders f_1 and f_2;

the number $boundlev$ fix the boundary level for the parallel process. Boundary level depends on characteristics of a computing cluster. For the given task $boundlev = \log_2 k$ because the graph of the considered algorithm will be a binary tree.

Let's describe a method go_down.

go_down(intervals, r, boundlev) {
$s = \lfloor (intervals[r] + intervals[r+1])/2 \rfloor$;
$intervals1 = intervals$; $r1 = r$;
$intervals2 = intervals$; $r2 = r$;
$intervals1[r+1] = s$;
$intervals2[r] = s$;
$if\ (s == intervals1[r] + 1)\ r1- = 2$;
$if\ (s == intervals2[r+1] - 1)\ r2- = 2$;
$boundlev - -$;
}

The parallel algorithm consists of $2\log_2 k$ levels. Let's describe operations which are fulfilled on processors at each level.

1) Processor 0.
$boundlev = \log_2 k$;
$intervals[\,] = numbOfMod()$;
$r = \ k == 1?\ 2(t+1) - 3 : 2(t+1) - 1$;
$go\_down(intervals2, r2, boundlev)$;
$send(A, intervals2, r2, boundlev, k/2)$;

i) $(i = 2, \ldots, \log_2 k)$. Processors $jk/2^{i-1}$, $j = 0, \ldots, 2^{i-1} - 1$.
$recv(A, intervals, r, boundlev, (j-1)k/2^{i-1})$ for odd j;
$go\_down(intervals, r, boundlev)$;
$send(A, intervals2, r2, boundlev, (2j+1)k/2^i)$;

$1 + \log_2 k$) Processors j, $j = 0, 1, \ldots, k-1$.
$recv(A, intervals, r, boundlev, j-1)$ for odd j;
$f = charPol(A, intervals, r)$;
$send(f, j-1)$ for odd j;

$i + \log_2 k$) $(i = 2, \ldots, \log_2 k)$. Processors $j2^i$, $j = 0, \ldots, k/2^i - 1$.
$recv(f_2, (2j-1)2^{i-2})$;
$f = recoveryNewton(f_1, f_2, r1, r2)$;
$send(f, (2j+1)2^i)$ for odd j;

Remark 4 *If a matrix* $A \in Z^{n \times n}$ *then* $intervals = [1, h]$.

5 Experiments with the parallel algorithm

Experiments that were hold with characteristic polynomials of dense matrices of a numbers and a polynomials were computed had been made. Elements of matrices got out in a random way. All numerical coefficients have the equal number of bits.

For estimation of efficiency of parallel algorithms we enter the concept of an efficiency. Let t_k be the computation time of the algorithm for the cluster with k processors. At transition

from the cluster with n processors to the cluster with k processors, $k > n$, the efficiency is equal 100%, When $t_n/t_k = k/n$. The efficiency is equal to zero, when $t_k = t_n$. To define an efficiency of computations at other values t_n/t_k we define the efficiency as the time function t_k.

Definition 1 *Efficiency of computations on k processors in comparison with computation on m processors is the function*

$$\alpha_{m,k} = \frac{t_m/t_k - 1}{k/m - 1} \cdot 100\%.$$

In experiments 1 and 2 we used two parallel algorithms (algorithm N and algorithm D). Algorithm D computes the characteristic polynomial of a matrix in a finite field with the help of Danilewsky's algorithm [7], algorithm N – with the help of an algorithm in the work [6].

Experiment 1 on a supercomputer MVS100k of Joint Supercomputer Center of the RAS were made [13].

In the experiment we used dense integer matrix. The size of a matrix is 1000×1000. The number of processors is from 16 to 512.

The time and the efficiency of computations are presented in the table 1.

Table 1

The time and the efficiency of computations with the help of algorithms N and D for matrices of an order 1000×1000 and $\log_2 a = 7$ bits

Quantity of processors	Algorithm N		Algorithm D	
	Time t_k, s	**Efficiency $\alpha_{16,k}$, %**	**Time t_k, s**	**Efficiency $\alpha_{16,k}$, %**
16	1849		1507	
32	921	100	764	97
64	562	76	386	96
100	522	48	364	59
127	500	38	355	46
128	310	70	226	80
175	267	59	220	58
255	239	45	167	53
256	166	67	127	72
350	162	49	122	54
400	113	64	89	66
512	113	49	83	55

Apparently from table 1 if quantity of processors are divisible by 2^p computation time extremely decreases where p – natural number.

Experiment 2 was hold with a cluster from 16 processors of Intel Xeon 3 GHz, 1 Gb, installed in a laboratory of algebraic calculations of the Tambov State University named after G.R. Derzhavin. In the experiment we used dense integer matrix. The size of a matrix is 400×400 and if a is a largest absolute value for coefficients of matrix then $\log_2 a = 20$ bits. The number of processors is from 2 to 16. The time and the efficiency of computations are presented in the table 2.

Table 2

The time and the efficiency of computation with the help of algorithms N and D for matrices of an order 400×400 and $\log_2 a = 20$ bits

Quantity of processors	Algorithm N		Algorithm D	
	Time t_k, s	**Efficiency $\alpha_{2,k}$, %**	**Time t_k, s**	**Efficiency $\alpha_{2,k}$, %**
2	2740		1648	
4	1369	100	816	102
6	1268	58	833	48
8	691	98	416	98
10	660	78	429	71
12	644	65	426	57
14	660	52	427	47
16	359	94	222	91

Experiments have shown that efficiency (Table 2) is in limits from 50 % to 98 %. The best efficiency is reached, when the number of processors is a degree of number 2.

In experiments 3 and 4 with polynomial matrices we used the parallel algorithm D which computes the characteristic polynomial in a finite field with the help of Danilewsky's algorithm [7]. These experiments were hold with a supercomputer MVS100k of Joint Super-Computer Center of the RAS.

In experiment 3 we used dense polynomial matrix of two variables: $s = [2, 2]$, $a = 10$ bits, $n = 50$. In experiment 4 we used dense polynomial matrix of two variables: $s = [1]$, $a = 10$ bits, $n = 400$.

The times and the efficiency of computations are presented in the table 3.

Table 3

The time and the efficiency of computations with the help of algorithms N and D for polynomial matrices of an order $n \times n$, $b = 10$ bits is largest absolute value for numerical coefficients of matrix elements, $m_1, \ldots, m_t$ is a hight degrees of variables $x_1, \ldots, x_t$

$n = 50$, $m_1 = 2$, $m_2 = 2$

Quantity of processors	**Time, s** t_k	**Efficiency, %** $\alpha_{1,k}$
1	16558	
2	8676	91
4	4548	88
8	2651	75
16	1626	61
32	1146	43
64	748	34
128	513	25
256	510	12

$n = 400$, $m_1 = 1$

Quantity of processors	**Time, s** t_k	**Efficiency, %** $\alpha_{1,k}$
16	14514	
32	8178	77
64	5101	61
128	2882	57
256	2046	40
512	1576	26
1024	1445	14
2048	1354	7
4096	1316	3

Experiments show that while increase in the number of processors increases the efficiency of calculations decreases. Then further multisequencing does not become favorable to some

number of processors. Transferred blocks become so small that transfer time comes close to computation time at boundary level. For example, for experiment 3 it is ineffective to use the considered parallel algorithm for computing the characteristic polynomials with usage of 256 and more processors as computation time does not decrease. The best computation time for polynomial matrices (the order 50×50, $b = 10$ bits is largest absolute value for numerical coefficients of matrix elements, $[2, 2]$ of highest degrees of variables x_1, x_2) on cluster MVS100й will be about 10 minutes, and for matrices (the order 400×400, $b = 10$, $[1]$ of the highest degrees of variables x_1) will be about 25 minutes (Table 3).

6 Conclusion

Parallel implementation of algorithms allows to compute characteristic polynomials for matrices of a big size. Therefore, it is important to construct effective parallel algorithms. Modular arithmetics allows to make it as calculations on each module independently upon each other. If algorithms based on the method of homomorphic images over a finite field use the best according to the number of operations algorithms for calculation of characteristic polynomials it is possible to obtain effective parallel algorithms.

There were developed the parallel programs which realize two algorithms (algorithm N and algorithm D) of computation of characteristic polynomials for numerical matrices and one algorithm of computation of characteristic polynomials for polynomial matrices of many variables . Algorithm N in a finite field uses the algorithm from the work [13] which has the best estimation of ring operations ($\sim 7/3n^3$). Algorithm D uses Danilewsky's algorithm [7] which has $\sim 2n^3$ operations in a finite field. Graphs of algorithms N and D are binary trees. Therefore it is effective to use the parallel computer which has 2^p processors. Really, experiments showed that the efficiency of computations is the greatest at transition from 2^p to 2^{p+1} processors, it is 75% – 94%. Experiments showed that computation time of characteristic polynomials of matrices by the algorithm D is 20-60% less, than on algorithm N.

Taking into account the obtained results of experiments in the ring of integers for calculation of characteristic polynomials of matrices in the ring of polynomials the algorithm which uses Danilewsky's algorithm in a finite field has been realized. Experiments show that at increase of the number of processors the efficiency of calculations decreases. If the number of processors increases, the number of transfers also increases, and the size of calculations at boundary level decreases. For some number of processors the sending time will be equal to computation time at boundary level, further parallelization is not effective. For characteristic polynomials computing for matrices of the size 50×50 over polynomials of two variables whose highest degrees equal 2 and the greatest absolute value of numerical coefficients has 10 bits, it is not effective to use the considered parallel algorithm on 128 and more processors. For matrices of the size of 400×400 over linear polynomials of one variable and 10 bits greatest absolute value of numerical coefficients – on 512 and more processors.

The considered algorithms for computing of characteristic polynomials for matrices over a ring of integers and over a ring of polynomials showed good scalability. It is supposed to realize algorithms over a finite field using Keller-Gehrig's algorithm [8] and Pernet-Storjohann algorithm (2007) [9], to compare them with algorithms already realized and to reveal the most effective algorithms.

References

1. *Le Verrier U.J.J.* Sur les variations séculaires des éléments elliptiques des sept planétes principales: Mercure, Venus, La Terre, Mars, Jupiter, Saturne et Uranus//J. de Mathématiques Pures et Appliquees. 1840. N. 4. P. 220-254.

2. *Faddeev D.K., Faddeeva V.N.* Computational methods of linear algebra. San Francisco: W.H. Freeman, 1963.

3. *Knut D.* The art of computing programming. V. 2. M., 1977.

4. *Seifullin T.R.* Computation of determinants, adjoint matrices, and characteristic polynomials without division//Cybernetics and Systems Analysis. 2003. V. 39, N. 6. P. 805-815.

5. *Malaschonok G.I.* A computation of the characteristic polynomial of an endomorphism of a free module//Zapiski Nauchnyh Seminarov POMI. 1999. V. 258. P. 101-114.

6. *Pereslavtseva O.N.* Method for computing of matrix characteristic polynomial//Tambov University Reports. Natural and Technical Sciences. 2008. V. 13. Issue 1. 2008. P. 131-133.

7. *Danilewsky A.M.* About numerical solution of a secular equation//Rec. Math. 1937. V. 2(44). N. 1. P. 169-172.

8. *Keller-Gehrig W.* Fast algorithms for the characteristic polynomial//Theoretical computer science. 1985. V. 36. P. 309–317.

9. *Pernet C., Storjohann A.* Faster algorithms for the characteristic polynomial//ISSAC. 2007. P. 307–314.

10. *Buchberger B., Collins G. E., Loos R.* Computer Algebra – Symbolic and Algebraic Computation. Vienna; New York: Springer-Verlag, 1982.

11. *Dumas J.-G., Pernet C., Wan Z.* Efficient Computation of the Characteristic Polynomial // ISSAC'05, July 24–27, 2005, Beijing, China, Beijing, 2005. P. 140-147.

12. *Pereslavtseva O.N.* Computation of characteristic polynomials for matrices over polynomial ring//International Conference Polynomial Computer Algebra. St. Petersburg, PDMI RAS, 2009. P. 35-39.

13. *Pereslavtseva O.N.* On the computation of characteristic polynomial coefficients//Numerical Methods and Programming. 2008. V. 9. P. 366-370. URL: http://num-meth.srcc.msu.ru/.

GRATITUDES: Supported by the Sci. Program Devel. Sci. Potent. High. School, RNP 2.1.1.1853.

Accepted for publication 7.06.2010.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ВЫЧИСЛЕНИЯ ХАРАКТЕРИСТИЧЕСКИХ ПОЛИНОМОВ МАТРИЦ, ОСНОВАННЫЙ НА МЕТОДЕ ГОМОМОРФНЫХ ОБРАЗОВ

© **Оксана Николаевна Переславцева**
Тамбовский государственный университет им. Г.Р. Державина, Интернациональная, 33, Тамбов, 392000, Россия, программист лаборатории алгебраических вычислений, e-mail: Pereclavtseva@rambler.ru

Ключевые слова: вычисление характеристических полиномов матриц; параллельный алгоритм; метод гомоморфных образов; кластер.

Предлагаются параллельные алгоритмы для вычисления характеристических полиномов целочисленных и полиномиальных матриц. Данные алгоритмы основаны на методе гомоморфных образов, примененном как к кольцу целых чисел, так и к кольцу полиномов многих переменных. Для применения метода гомоморфных образов находится верхняя оценка числовых коэффициентов характеристического полинома. Обсуждаются результаты экспериментов с параллельными алгоритмами, проведенных на кластере МВС-100К в МСЦ РАН.

UDC 519.688

OUT-OF-CORE PARALLEL POLYNOMIAL ARITHMETIC

© **Alexey Gennadievich Pozdnikin**
Tambov State University named after G.R. Derzhavin, Internatsionalnaya, 33, Tambov, 392000 Russia, Post-graduate Student of Computer and Mathematical Modeling Department, e-mail: pozdalex@mail.ru

Key words: polynomials on the external data carrier; polynomial arithmetic; the parallel algorithm of multiplication.

This paper presents the description of structure of polynomials on the external data carrier. The algorithms for addition and parallel multiplication of polynomials are scrutinized. The results of experiments conducted with parallel multiplication of polynomials on cluster are given.

1 Introduction

Polynomials are the main objects in symbolic computation [1]. The effectiveness of computer algebra system depends on the effectiveness of polynomial procedures.

Symbolic computations are characterized as problems of high computational complexity. Therefore, it is necessary to develop parallel algorithms and conducting calculations on multiprocessor computer systems.

In the articles [3], [4] there is information about parallel polynomial algorithms. Traditional systems of computer algebra, such as Mathematica, can operate on polynomials, that do go into RAM. However, these systems are unsuitable for operation with large polynomials, which need more memory and cannot be written into RAM. Therefore, providing operating on such polynomials is one of the primary tasks of parallel computer algebra.

One of such systems that can operate on so large mathematical expressions is «FORM». It is a system for symbolic manipulation of algebraic expressions specialized in handling with very large expressions of millions of terms in an efficient and reliable way [5].

In the article [6] the representation of the large polynomials is discussed, the algorithms realising an implementation of main arithmetical operations are considered and the results of the experiments are also presented there.

This article describes the structure of the polynomial, which is stored on the external data carrier. Algorithms of addition and parallel multiplication of such polynomials are considered there.

You may also get acquainted with the results of experiments which were carried out on operation of parallel multiplication of polynomials on cluster of JSC RAS. It is presented in graphics.

2 The structure of polynomials on the external data carrier

We used two one-dimensional arrays to store one polynomial. The first array stores only the nonzero coefficients of a polynomial and the second array stores the degrees of each variables. If there are «var» variables in a polynomial, the second array contains «var» times more elements than the first. Monomials in the polynomial are stored in reverse lexicographical order. This order is accepted, that arithmetic algorithms with polynomials worked faster. You can learn about other structures of polynomials in article [7].

This polynomial will be stored on external data carrier in two files. Monomials of a polynomial in the form of arrays bytes will be saved in one file. The second file will contain the type of coefficient, i.e. the set of whole or rational numbers which polynomial coefficients are taken from. Then, the number of variables of the polynomial (vars), the total number of nonzero monomials in the polynomial and an array of integers will be written in the second file. The array of integers contains the information about number of bytes, which each monomial of the polynomial occupies on a hard disk. We will call such polynomial the file polynomial, which are stored in external memory.

We should be able to operate with file polynomials and to send them between cores. For this purpose, we will operate small fragments of file polynomials, which can be located in RAM.

3 Addition of file polynomials

Operation of addition of file polynomials is implemented in the form of consecutive algorithm.

This algorithm consists of three main parts:

1. We compare variables in file polynomials. If one of the file polynomials has more the number of variables, the monomials, that contain these older variables will be recorded in the resulting file. Otherwise, we go to the step 2.

2. We compare exponents of variables in each monomial. If exponents of variables in the monomials are equal, the coefficients are added and a new monomial with the same degrees is recorded in the file. If the sum of coefficients is equal to zero, then the monomial at this degree is not written into file. If the exponents of variables in one of the monomials will be greater, then this monomial can be written in the file, and the smallest one is compared with the following one. Transition to the next step will be done when the file polynomials will be read to the end.

3. We read and write into the resulting file of the remaining monomials of one of polynomials.

Let

$$p_1 = 9x^2y^2 + 4x^2y - 8xy + x - 6,$$

$$p_2 = -8x^2y^2z^3 - x^2y^2z^2 - 4x^2y - 5x^3 + 3xy.$$

We consider example of addition $p_(1)$ and $p_(2)$.

Step 1. We write into the file: $-8x^(2)y^(2)z^(3) - x^(2)y^(2)z^(2)$.

Step 2. We write into the file: $9x^(2)y^(2) - 5x^(3) - 5xy$.

Step 3. We write into the file: $x - 6$.

As a result, we obtain the sum of two polynomials in form of a sorted file polynomial.

If we view $p_(1) + p_(2)$ then the following will be written in the file:

$$p = -8x^2y^2z^3 - x^2y^2z^2 + 9x^2y^2 - 5x^3 - 5xy + x - 6.$$

4 The parallel algorithm of multiplication of file polynomials

The procedure of multiplication of file polynomials is recursive. Dichotomous division of polynomials on the part present a basis of the recursive algorithm.

The condition is a way of the exit out of the recursion, if it is satisfied, then the multiplication of individual parts of polynomials can be made in memory of the given size.

The value of free RAM is set by a variable $freeMemory$. Procedure $getMemForMul$ estimates size of the memory that may be required to multiplication of two polynomials or their parts. The result returned by the procedure $getMemForMul$ is compared to the variable $freeMemory$.

The binary tree is the graph of the recursive algorithm. The multiplication of parts of polynomials is performed on its leaves.

The interval with numbers of free cores is set in root node. The parallel algorithm of splitting of polynomials by parts is accompanied by splitting the interval with numbers of cores. If set of free cores is empty and multiplication of parts of polynomials cannot be done in memory, the consecutive recursive algorithm on one core will be caused.

We consider a parallel algorithm for multiplication of file polynomials A and B. Algorithm's graph is presented at Figure 1.

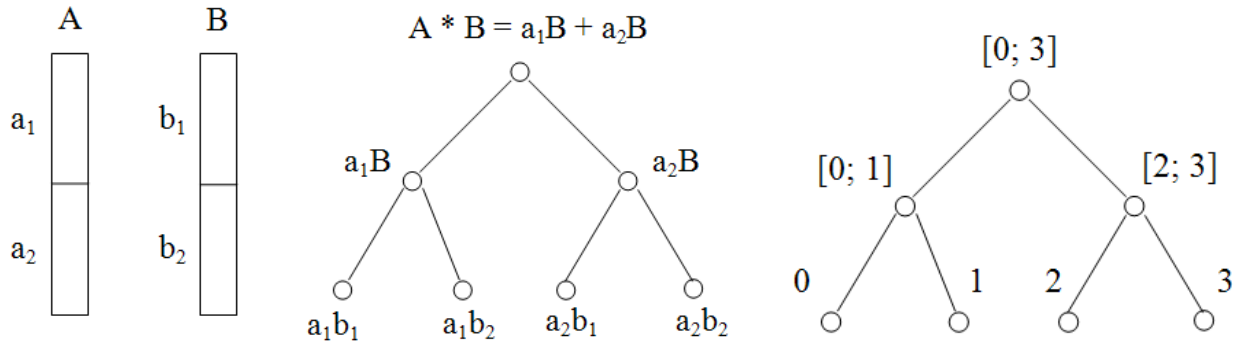

Fig. 1. The graph of parallel algorithm of multiplication of polynomials A and B and distribution of four cores to nodes of the tree

Let $A = (a_1 + a_2)$ and $B = (b_1 + b_2)$ be two file polynomial that we want to multiply. The product can be found as the sum of four items: $a_1 * b_1 + a_1 * b_2 + a_2 * b_1 + a_2 * b_2$. The calculation of each of the four items can be executed on a separate core.

We choose greater polynomial and splitted it by two parts. Parts should occupy in the memory an equal amount of bytes. Let the polynomial $A > B$, ie A occupies more memory than B. On the first step we divide a polynomial A into two parts a_1 and a_2, ie $A = a_1 + a_2$. The interval with numbers of cores $[0, 3]$ will be divided into two intervals $[0, 1]$ and $[2, 3]$. We received two nodes $a_1 * B$ and $a_2 * B$. These operations cannot be executed in RAM, therefore division of polynomials into parts will be continued.

We choose greater of polynomials a_1 and a_2. Let $B > a_1$ and $B > a_2$. Then we divide B into parts b_1, b_2 and calculate products $a_1 * b_1$, $a_1 * b_2$, $a_2 * b_1$, $a_2 * b_2$ on each core.

Let we have reached leaf nodes if multiplication is possible to execute in RAM on cores 0, 1, 2, 3, accordingly.

During the sending the calculated fragments back to the root, their addition will be done: $a_1 * b_1 + a_1 * b_2 = a_1 * B$ and $a_2 * b_1 + a_2 * b_2 = a_2 * B$. The result of multiplication will be the sum $a_1 * B + a_2 * B = A * B$, calculated at the root.

We consider the program code of procedures for parallel multiplication of the file polynomials, implemented on language Java.

We introduce the following designation:

Polynom – is a type of polynomial, which is stored in memory.

FPolynom – is a type of file polynomial.

Subset – is a set of numbers of available cores.

BasePolynomDir – is a class that is used to create the directory where the file will be written polynomials.

By default it is a directory "*/tmp/fpolynoms/*" in operating systems Linux and "*C : \temp\fpolynoms*" in Windows.

In algorithm of parallel multiplication of file polynomials the procedures are used:

1) *Polynom mulS*(*Polynom pol*2). The procedure multiply polynomials in RAM.

2) *Polynom toPolynom*(*long skipBytes*, *long bytes*). The procedure reads a part of the file polynomial and writes it into RAM. Parametres: skipBytes — quantity of bytes which will be skipped, bytes - bytes quantity which will be read. Result is a polynomial in RAM.

3) *FPolynom toFPolynom*(*File filename*, *Element itsCoeffOne*). The procedure reads a polynomial from memory and writes down on a hard disk, at the specified path filename. itsCoeffOne - is a unit in the field of the coefficients of the polynomials. The result is a polynomial, written in external memory.

4) *long getMemForMul*(*FPolynom fpol*1, *FPolynom fpol*2, *long s*1, *long n*1, *long s*2, *long n*2). The procedure returns the number of bytes which can be received as a result of multiplication of parts of polynomials *fpol*1 and *fpol*2. *s*1, *s*2 is a bytes which will be skipped in the polynomials *fpol*1 and *fpol*2. *n*1, *n*2 is a bytes which will be read in the polynomials *fpol*1 and *fpol*2.

5) *long getByteLength*(). The procedure returns the size of memory in bytes that the file polynomial occupied.

6) *Subset*[] *divideOnParts*(*int n*). The procedure splits an interval into n parts and returns an array of intervals.

7) *long middlePolynom*(*long skipBytes*, *long middle*). The procedure returns the number of bytes approximately equal to half of a memory size which occupies a part of a file polynomial. *skipBytes* is the bytes needs to be skiped a file polynomial, *middle* - is the middle of a part of the file polynomial.

8) *Ssend*(*Object obj*, *int proc*, *int tag*). The procedure sends an object *obj*, to the core with number of *proc*, and of tag is the *tag*.

9) *Recv*(*int objType*, *intproc*, *int tag*). The procedure receives a object *obj*, from the core with number of proc, and of tag is the *tag*.

10) *SendFPolynom*(*FPolynom pol*, *long skipBytes*, *long numbytes*, *intproc*).
The procedure sends numbytes bytes of a file polynomial pol to the core with number proc. *skipBytes* of bytes will be skiped from the file beginning.

11) *RecvFPolynom*(*File dir*, *int proc*) - The procedure receives a file polynomial from the core with number *proc* and writes down his on a disk in the directory *dir*.

12) *add*(*String dir*1, *String dir*2, *File fdir*). The procedure adds file polynomials which are in the directories *dir*1, *dir*2 and writes the result in *fdir*.

The program code of procedure of multiplication of file polynomials can be seen in Fig. 2.

```
public static FPolynom multiply(FPolynom fpol1,
FPolynom fpol2, File fres) throws Exception{
   int myrank = MPI.COMM_WORLD.Rank();
   if(myrank == 0){
      int size = MPI.COMM_WORLD.Size();
      Subset procs = new Subset(new int[]{0,size-1});
      multiplyRec(fpol1, fpol2, 0, fpol1.getByteLength(),
      0, fpol2.getByteLength(), fres, procs, 0)}
   else{
      Status st = MPI.COMM_WORLD.Probe(MPI.ANY_SOURCE, MPI.ANY_TAG);
      if(st.tag==tag_true){
         int parent = (Integer)LLP.Recv(LLP.INT_TYPE, MPI.ANY_SOURCE, tag_true);
         BasePolynomDir dir = new BasePolynomDir();
         File f1 = new File(dir.createPolynomDir("proc"+myrank), "p1");
         File f2 = new File(dir.createPolynomDir("proc"+myrank), "p2");
         File f3 = new File(dir.createPolynomDir("proc"+myrank), "p3");
         int[] arr = (int[])LLP.Recv(LLP.INT_ARRAY_TYPE,
         parent, tag_proc);
         Subset process = new Subset(arr);
         LLP.RecvFPolynom(f1, parent);
         LLP.RecvFPolynom(f2, parent);
         FPolynom p1 = new FPolynom(f1);
         FPolynom p2 = new FPolynom(f2);
         multiplyRec(p1, p2, 0, p1.getByteLength(),
         0, p2.getByteLength(), f3, process, myrank);
         LLP.SendFPolynom(new FPolynom(f3), 0,
         f3.length(), parent); }}
   return new FPolynom(fres);}
```

Fig. 2. The code of procedure of multiplication of file polynomials

Procedure multiply receives on an input two file polynoms *fpol*1, *fpol*2 and a directory fres in which the result of multiplication will be written down.

The procedure *Size*() determines the number of core and puts his in the variable myrank.

On the core with number zero (*myrank* =0), the variable size accepts value of total number of cores, numbers of cores will be contain in an interval procs from 0 to $size - 1$.

On zero core recursive procedure of multiplication multiply parts of polynoms *fpol*1, *fpol*2 is started. The remaining cores, with numbers not equal to zero, waiting for a message with tag equal to the *tag_true*.

When a message with *tag_true* will come, then the number of core from which it came, the interval with numbers of available cores, and two polynomial will be received.

```
private static FPolynom multiplyRec( FPolynom fpol1, FPolynom fpol2,
long skip1, long length1, long skip2, long length2,
File fres, Subset proc, int myrank) throws Exception{
File bufres = fres;
FPolynom result = new FPolynom(fres);
String namedirA, namedirB;
int l1 = end1-st1, l2 = end2-st2;
if(getMemForMul(fpol1, fpol2, skip1, length1, skip2, length2)<freeMemory){
    fpol1.toPolynom(skip1, length1).mulS(
    fpol2.toPolynom(skip2, length2)).toFPolynom(fres, itsCoeffOne);
    if(proc.cardinalNumber()>1)
        for(int i=1; i<proc.cardinalNumber(); i++)
            LLP.Isend(new Integer(0), proc.toArray()[i],
            tag_false);}
        else{ long s1=skip1, s2=skip2, e1=length1, e2=length2,
            s11=0, s22=0, e11=e1, e22=e2;
            Subset[] process;
            if(proc.cardinalNumber()>1){ process = new Subset[2];
                process = proc.divideOnParts(2);
                LLP.Ssend(new Integer(myrank), process[1].toArray()[0], tag_true);
                LLP.Ssend(process[1].toArray(), process[1].toArray()[0], tag_proc);
                if(length1>=length2){ e1 = fpol1.middlePolynom(skip1, e1/2);
                    LLP.SendFPolynom(fpol1, s1, e1, process[1].toArray()[0]);
                    LLP.SendFPolynom(fpol2, s2, e2, process[1].toArray()[0]);
                    s11 = e1+skip1; e11 = length1-e1; s22=skip2;
                } else{ LLP.SendFPolynom(fpol1, s1, e1, process[1].toArray()[0]);
                    e2 = fpol2.middlePolynom(skip2, e2/2);
                    LLP.SendFPolynom(fpol2, s2, e2, process[1].toArray()[0]);
                    s22 = e2+skip2; e22 = length2-e2; s11=skip1;}}
            else{ process = new Subset[1];
                process[0] = proc;
                if(length1>=length2){ e1 = fpol1.middlePolynom(skip1, e1/2);
                    s11=e1+skip1; e11=length1-e1; s22=skip1;
                }else{ e2 = fpol2.middlePolynom(skip2, e2/2);
                    s22=e2+skip2; e22=length2-e2; s11=skip1;}}
                    fileA = fres.getAbsolutePath()+"a";
                    bufres = new File(fileA);
                    multiplyRec(fpol1, fpol2, s11, e11, s22, e22, bufres, process[0], myrank);
                    fileB = fres.getAbsolutePath()+"b";
                    bufres = new File(fileB);
                    if(proc.cardinalNumber()>1){ LLP.RecvFPolynom(bufres, process[1].toArray()[0]);
                    else{ multiplyRec(fpol1, fpol2, s1, e1, s2, e2, bufres, process[0], myrank); }
                    FPolynom.add(fileA, fileB, fres);
            return result;
    }
```

Fig. 3. The code the recursive procedure of the multiplication of parts of file polynomials

Recursive procedure of multiplication of the received polynomials will be caused. The result of multiplication is sent back, to the core from which polynomials have been received. If the message with tag tag_true is not received by the core then he is remains not used.

4.1 The recursive procedure of multiplication of parts of file polynomials

The recursive procedure of multiplication will have following arguments:

1) two file polynomials $fpol1$ and $fpol2$;
2) number of bytes which is necessary to skip in the file polynomial $fpol1$;
3) number of bytes which is necessary to read from the file polynomial $fpol1$;
4) number of bytes which is necessary to skip in the file polynomial $fpol2$;
5) number of bytes which is necessary to read from the file polynomial $fpol2$;
6) the directory fres in which the result of multiplication will be written down;
7) the interval of procs which contains numbers of cores;
8) the number of node on which procedure is caused.

The program code the recursive procedure of multiplication of file polynomials can be seen in Figure 3.

After initialization of some variables, there is a condition check. The result of multiplication of polynomials or their parts must will be located in RAM, ie the volume $freeMemory$. If this condition is satisfied, then they are multiplied in memory, the result is returned. All untapped cores sent a message with tag equal tag_false, denoting the end of the operation.

If function returns value exceeding freeMemory, then greater of the polynomials will be splitted on two parts. One of pairs of parts from the file polynomials remains on one node, and the second pair is sent to another core. Division of polynomials into parts will be will proceed until product of these parts will be located in RAM in volume freeMemory. After parts of polynomials have been multiplied, product will be sent the core from which they have been received. The core will calculate the sum of the received polynomials. On zero core last operation of addition of polynomials will be made.

5 Experiments

The program complex has been developed. The experiments were conducted on the cluster of $MVS-100K$ in the MSC Russian Academy of Sciences. At experiments we used polynomials of two variables, received in a random way, with coefficients not greater than 10^3 by absolute value and quantity of monomials $25*10^4$. For parallel algorithm it is accepted that free RAM, ie freeMemory it is equal 32 Mb.

Let:

T_0 – The time of calculations on n cores;

T_k – The time of calculations on k cores;

$k > n$.

The speedup of calculations at transition from n cores to k cores will be assumed by the formula $a(T_k) = (1 - T_0/T_k)/(1 - k/n) * 100$. The speedup is measured in percents. In this experiment $n = 8$. The results of experiments are presented in Tables 1 and 2.

Table 1

The table of values of run–time of operation of multiplication of polynomials on n cores and speedups of calculations on n cores in comparison with calculations on one core. One core is used on each node

number of cores	time, sec	efficiency, %
8	2644	–
16	1719	53,8
32	1176	41,6
64	785	33,8
128	577	23,9

Table 2

The table of values of run–time of operation of multiplication of polynomials on n cores and speedups of calculations on n cores in comparison with calculations on one core. Eight core is used on each node

number of cores	time, sec	efficiency, %
8	3713	–
16	2578	44,0
32	1688	40,0
64	1009	38,3
128	716	27,9

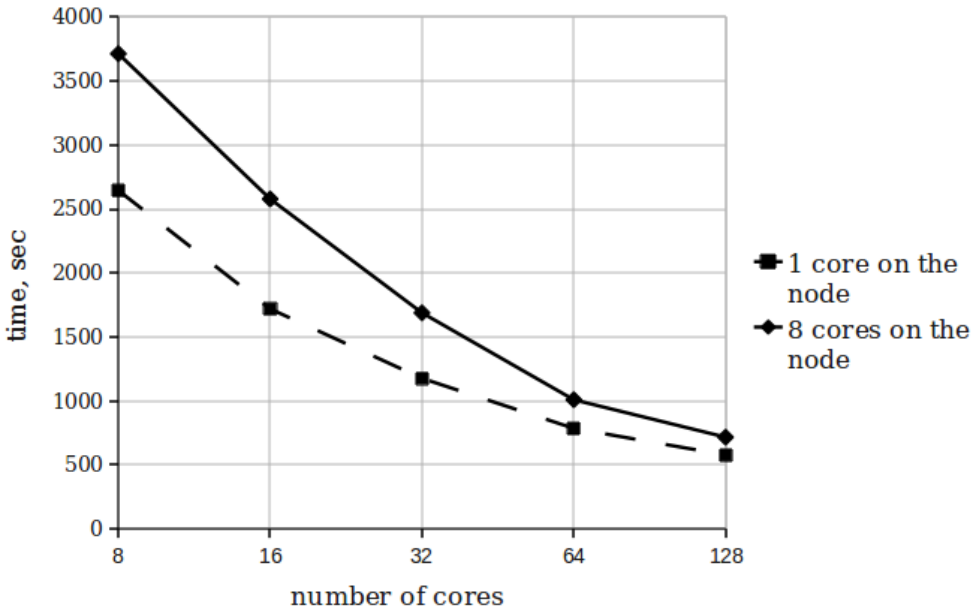

Fig. 4. The graph of dependence of run–time of operation of multiplication of polynomials from number of cores

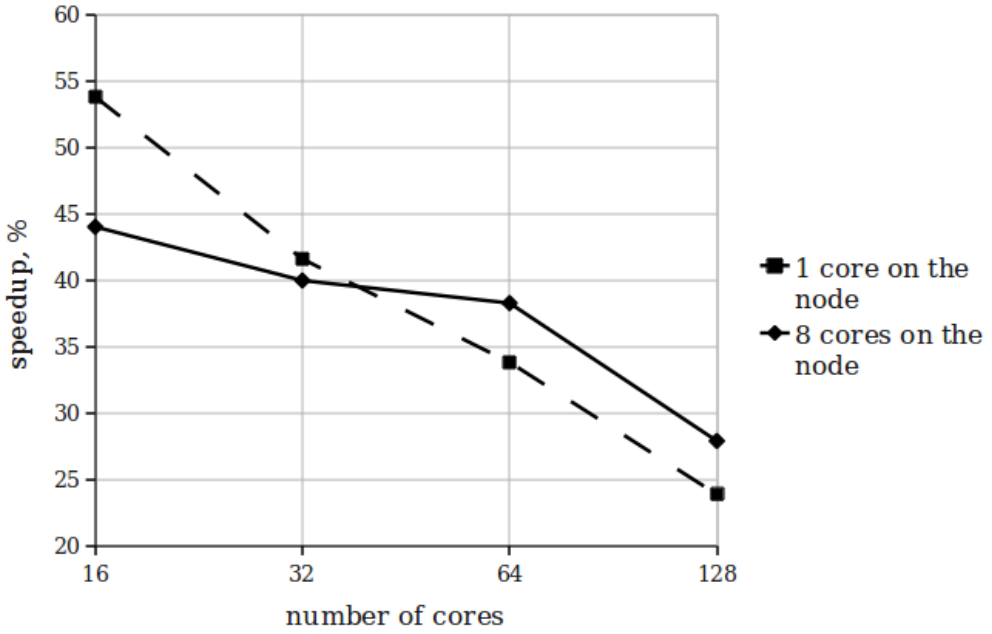

Fig. 5. Efficiency of run-time of operation of multiplication on k–core cluster, in comparison with calculations on n–core cluster

6 Conclusions

We can see on the graph in Figure 4 that with an increase of number of cores, run time of operation of multiplication decreases.

There are 8 cores on each node on the cluster MBC–100K. If we use single core on 8 nodes, operation will be executed faster than when using 8 cores on one node. Because 8 cores on one node use one hard disk. When we set the task for 8 nodes, and use only 1 core on the node, instead of 8 possible, the most of cores in this case will not work. Therefore, use of all cores on the node is more profitable and is more economical.

On the graph of Figure 5 we can see that the speedup time of the operation of multiplication decreases with increasing number of cores. If we continue to increase quantity of cores, speedup becomes close to zero. In the above example, we not used more than 128 cores when speedup of calculations in comparison with 8 cores will be less than 30 %.

The realised parallel algorithm of multiplication of file polynomials has shown the efficiency and can be applied dealing with problems which use multiplication of polynomials of the big sizes.

References

1. *Pankratiev E.V.* Elements of computer algebra//Internet university of an information technology. Laboratory of knowledge. 2007. P. 248.

2. *Malaschonok G.I., Avetisan A.I., Valeev U.D., Zuev M.S.* Parallel algorithmes of computer algebra // Proceeding of the institute of system programming. 2004. V. 8. Issue 2. P. 169-180. (Russian).

3. *Valeev U.D., Malaschonok G.I.* On the forms of polynomials for parallel calculations// Tambov University Reports. Natural and Technical Sciences. 2004. V. 9. N. 1. P. 149-150. (Russian).

4. *Malaschonok G.I., Valeev Y.D.* Parallel polynomial recursive algorithms// International conference Polynomial Computer Algebra. St. Petersburg: PDMI RAS, 2008. P. 41-45. (Russian).

5. *Fliegner D., Retey A., Vermaseren J.A.M.* Parallelizing the symbolic manipulation program FORM. URL: http://arXiv.org/abs/hep-ph/0007221.

6. *Pozdnikin A.G.* File polynomials// Tambov University Reports. Natural and Technical Sciences. 2009. V. 14. Issue 4. P.783-785.

7. *Yan T.* The Geobucket Data Structure for Polynomials// J. Symbolic Computation. 1998. P. 285-293.

GRATITUDES: Supported by the Sci. Program Devel. Sci. Potent. High. School, RNP 2.1.1.1853.

Accepted for publication 7.06.2010.

ПАРАЛЛЕЛЬНЫЕ ПОЛИНОМИАЛЬНЫЕ ВЫЧИСЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ ВНЕШНЕЙ ПАМЯТИ

© **Алексей Геннадьевич Поздникин**

Тамбовский государственный университет им. Г.Р. Державина, Интернациональная, 33, Тамбов, 392000, Россия, аспирант кафедры компьютерного и математического моделирования, e-mail: pozdnikin@mail.ru

Ключевые слова: полином на внешнем носителе; умножение полиномов; параллельный алгоритм.

В статье приводится описание строения полинома, который хранится на внешнем носителе. Рассматриваются алгоритмы сложения и параллельного умножения таких полиномов. Приводятся результаты экспериментов, которые проводились на кластере.

UDC 517.44+519.63

PARALLEL COMPUTING FOR FOURIER TRANSFORM WITH DISCONTINUOUS COEFFICIENTS

© **Oleg Emanuilovich Yaremko**
Penza State Pedagogic University named after V.G. Belinsky, Lermontova, 37, Penza, 440026, Russia, Candidate of Physics and Mathematics, Professor, Head of Mathematical Analysis Department, e-mail: yaremki@yandex.ru

© **Natalia Nikolaevna Yaremko**
Penza State Pedagogic University named after V.G. Belinsky, Lermontova, 37, Penza, 440026, Russia, Candidate of Physics and Mathematics, Associate Professor of Mathematical Analysis Department, e-mail: yaremki@yandex.ru

Key words: Fourier transform with dividing points; direct Cauchy problem; inverse Cauchy problem; heat conduction equation.

A parallel computing algorithm is researched in the article. It allows finding a solution for the direct and inverse problem of the temperature field structure in semi-infinite piece-homogeneous rod. The Fourier transform method with dividing points provides the parallel computing for the solution of the specified problems.

1 Introduction

Uflyand Y.S. proposed the Fourier transforms with discontinuous coefficients in the '70s of 20th century [1]. This theory was further developed in M.P. Lenyuk's works [2]. Multidimensional case is considered by V.A.Il'in see, [3]. Vector case is researched in O.E Yaremko's works [4]. The Fourier transforms with discontinuous coefficients are applied for the direct and inverse problems of mathematical physics.

The integral Fourier transforms are made in the following way. Let $u(x,\lambda)$ and $u^*(\xi,\lambda)$ be respective solutions for the Sturm-Liouville problem and dual problem for the Fourier's operator

$$B=\sum_{j=1}^{n}\theta(x-l_{j-1})\,\theta(l_j-x)\,A_j^2\frac{d^2}{dx^2}\;+\theta(x-l_n)\,A_{n+1}^2\frac{d^2}{dx^2}$$

on the real semi-axis with dividing points $0\leqslant l_0<l_1<\ldots<l_n<l_{n+1}=\infty$, where A_m- square matrices of the size $p\times p$, $\theta(x)-$ unit step function.

Expansion theorem into own functions of the Fourier's operator is proved, therein[3].

Theorem 1 *(Expansion theorem).Let's suppose vector- function $f(x)$ as defined, continuous, absolutely integrated on the set I_n^+ and has the limited variation on the set I_n^+. Decomposition formula takes the form*

$$f(x)=\frac{2}{\pi}\int_0^\infty \lambda u(x,\lambda)\left(\int_{l_0}^\infty u^*(\xi,\lambda)\,f(\xi)\,d\xi\right)d\lambda, x\in I_n^+,$$

where

$$I_n^+ = \left\{ x : \bigcup\nolimits_{j=1}^{n+1} (l_{j-1}, l_j) \right\}.$$

The direct F_{n+} and inverse F_{n+}^{-1} integral Fourier transforms with n dividing points are defined in the expansion theorem

$$F_{n+}[f](\lambda) = \int_{l_0}^{\infty} u^*(\xi, \lambda)\ f(\xi)\, d\xi \equiv \bar{f}(\lambda), \tag{1}$$

$$F_{n+}^{-1}\left[\bar{f}\right](x) \equiv \frac{2}{\pi} \int_0^{\infty} \lambda u(x, \lambda)\ \bar{f}(\lambda)\, d\lambda = f(x). \tag{2}$$

In the specific case the direct F_{n+} and inverse F_{n+}^{-1} transforms turn into the $cos-$ and $sin-$ transforms.

Modern mathematical models often lead to necessity of solution of differential equation systems for partial derivatives. Problems with piecewise constant coefficients arise at modeling processes of the multilayered environments. The vector integral Fourier transforms on the real semi-axis with dividing points are necessary mathematical methods for solution of such systems.

Some authors applied the Laplace transform [7] for solution of the specified problems. Solution of the problem is expressed by means of line integral even in a scalar case. The calculation of line integral is quite a difficult task and it requires appropriate skills.Actually the arising difficulties are insuperable in the vector case.

FFT method (Fast Fourier Transform) is realized in [8] for the Poisson equation in a circle. Fast algorithms of solution of direct and inverse problems for the vector heat conductivity equation with piecewise constant coefficients are constructed in the sec.2 and in the sec.3 respectively.

The purpose of this article is to develop a method fast singular vector Fourier transform. The offered method

-is an alternative to a classical net method;

-gives the standard engineering of solution of direct and inverse problems of mathematical physics with piecewise constant coefficients;

-admits the parallel computing processes.

Vector integral transforms of $cos-$ and $sin-$ types are constructed in sec.1.

Parallel computing processes of solution in direct problem of vector heat conductivity equation is shown in the sec.2 . Method of singular vector integral Fourier transforms is applied for solution of the inverse vector heat conductivity problem. It is well known that this problem is ill-posed one. Inverse heat conduction problem has been researched by many authors, see [9]. Iterative algorithm is proposed for solving of inverse problem in the sec.3.

2 Sturm-Liouville Problem

The Sturm-Liouville problem is to define the nontrivial solution of the mixed boundary value problem for the ordinary differential equation

$$\left(\frac{d^2}{dx^2} + q_m^2\right) u_m = 0,\ \ q_m^2 = A_m^{-2}\lambda^2,\ \ m = \overline{1, n+1}; x \in I_n^+ \tag{3}$$

by boundary conditions

$$\left(\alpha_{11}^0 \frac{d}{dx} + \beta_{11}^0\right) u_1 \Big|_{x=l_0} = 0, \quad \|u_{n+1}\| \,|_{x=\infty} < \infty \tag{4}$$

and contact conditions in dividing points

$$\left(\alpha_{j1}^k \frac{d}{dx} + \beta_{j1}^k\right) u_k = \left(\alpha_{j2}^k \frac{d}{dx} + \beta_{j2}^k\right) u_{k+1}, x = l_k, \;\; k = \overline{1,n}, \;\; j = 1,2. \tag{5}$$

Here - u_m vector- function of size $p \times 1$, $A_m, \alpha_{ji}^k -$ square matrices of the size $p \times p$.
Let us set

$$\varphi_{n+1}(x) = \exp(q_{n+1} x i); \;\; \psi_{n+1}(x) = \exp(-q_{n+1} x i); \;\; q_{n+1} = A_{n+1}^{-1}\lambda.$$

Let us define the other n-function pairs (φ_k, ψ_k), $k = \overline{1,n}$ as sequentially inductive relations

$$\left[\alpha_{j1}^k \frac{d}{dx} + \beta_{j1}^k\right](\varphi_k, \psi_k) = \left[\alpha_{j2}^k \frac{d}{dx} + \beta_{j2}^k\right](\varphi_{k+1}, \psi_{k+1}), \;\; k = \overline{1,n}, \;\; j = \overline{1,2}.$$

Also let us designate

$$\overset{0}{\underset{1}{\varphi}}(\lambda) = \left[\alpha_{11}^0 \tfrac{d}{dx} + \beta_{11}^0\right] \varphi_1(x,\lambda)\Big|_{x=l_0}, \quad \overset{0}{\underset{1}{\psi}}(\lambda) = \left[\alpha_{11}^0 \tfrac{d}{dx} + \beta_{11}^0\right] \psi_1(x,\lambda)\Big|_{x=l_0},$$

$$\Omega_k = \begin{pmatrix} \varphi_k & \psi_k \\ \varphi_k^{/} & \psi_k^{/} \end{pmatrix}.$$

Condition of unrestricted solvability of the problem (3) - (5) we will consider fulfilled further

$$\det \overset{0}{\varphi_1}(\lambda) \neq 0, \lambda \in (0,\infty).$$

For the Sturm-Liouville problem (3) - (5) by means of unit step function $\theta(x)$ we construct an appropriate spectral function $u(x,\lambda)$:

$$u(x,\lambda) = \sum_{j=1}^{n} \theta(x - l_{j-1})\, \theta(l_j - x)\, u_j(x,\lambda) + \theta(x - l_n)\, u_{n+1}(x,\lambda), \tag{6}$$

where

$$u_j(x,\lambda) = \varphi_j(x,\lambda) \overset{0}{\varphi_1^{-1}} - \psi_j(x,\lambda) \overset{0}{\psi_1^{-1}}.$$

The spectral function of the dual to the Sturm-Liouville problem (3) - (5) takes the following form

$$u^*(x,\beta) = \sum_{j=1}^{n} \theta(x - l_{j-1})\, \theta(l_j - x)\, u_j^*(x,\beta) + \theta(x - l_n)\, u_{n+1}^*(x,\beta), \tag{7}$$

where

$$u_j^*(x,\beta) = \left(\overset{0}{\underset{1}{\varphi}}(\beta), \; \overset{0}{\underset{1}{\psi}}(\beta)\right) \Omega_j^{-1}(x,\beta) \begin{pmatrix} 0 \\ E \end{pmatrix} A_j^2, \quad j = \overline{1,n+1}.$$

The direct F_{n+} and inverse F_{n+}^{-1} Fourier transforms on the real semi-axis [3] with n dividing points take the form (1), (2), where

$$f(x) = \sum_{k=1}^{n} \theta(l_k - x)\, \theta(x - l_{k-1})\, f_k(x) + \theta(x - l_n)\, f_{n+1}(x).$$

3 Vector Heat Conductivity Equation

The structure of a non-stationary temperature field of semi-infinite non-homogeneous rod takes the form [3]

$$v_i(t,x) = \int_0^\infty \exp\left(-\lambda^2 t\right) u_i(t,x)\, F(\lambda)\, d\lambda, \quad t > 0, \quad l_{i-1} < x < l_j \tag{8}$$

$$F(\lambda) = \sum_{j=1}^{n} F_j(\lambda), F_j(\lambda) = \int_{l_{j-1}}^{l_j} u_j^*(\xi,\lambda)\, f_j(\lambda)\, d\lambda,$$

where $f_j(\xi)$-is an initial distribution of temperature in j-a layer , $v_i(t,x)$-is a temperature distribution in i-a layer at the moment t.

First step. The processor P_j calculates the j component of the eigenfunction $u_j(\lambda,x)$ and the spectral function $F_j(\lambda)$.

Second step. Processors $P_1,\ldots,P_n$ transmit data to the processor P_i. Processor P_i calculates the values of temperature in the i-layer according to the formula (8) .

4 Inverse Vector Heat Conductivity Equation

Let's find the solution of the inverse heat conduction problem.The problem is to define initial distribution of the temperature field $f(x)$ in the equation (8) according to the known distribution $v(\tau,x)$ in a moment of time τ. Applying Fourier transform (1) - (2) in the equation (8), we obtain:

$$K(\lambda)\,\bar{f}(\lambda) = \bar{v}(\lambda),$$

where $K(\lambda) = \exp(-\lambda^2\tau)$.

We introduce the grid of nodes: $\lambda_k = -A + \frac{kA}{N}$, $k = 0,\ 1,\ldots,\ 2N$, were A –is quite a large number. Let's consider the iterative process [6]:

$$\bar{f}_{n+1}(\lambda_k) = \bar{f}_n(\lambda_k) - \gamma_k\left(K(\lambda_k)\,\bar{f}_n(\lambda_k) - \bar{v}(\lambda_k)\right), \tag{9}$$

$k = 0,\ 1,\ldots,\ 2N, \quad n = 0,\ 1,\ldots,$ where γ_k meets the following conditions:

$$q_k = |1 - \gamma_k K(\lambda_k)| \leqslant \frac{1}{2}, \quad k = 0,\ 1,\ldots,\ 2N.$$

It is always possible with this kind of function. We can show that for each

k $(k = 0,\ 1,\ldots,\ 2N)$ the speed converge iterations are Aq_k^n .

Then we find the function $f(x)$ on the quadrature formula using the inverse Fourier transform.

First step. The processor P_j calculates the j component of the spectral function $\overline{v}_j(\lambda)$.

Second step. Processors $P_1,\ldots,P_n$ transmit function values $\overline{v}_j(\lambda)$ to the processor P_0. Processor P_0 performs an iterative process according to the formula (9) .

Third step. Processors $P_0,P_1,\ldots,P_n$ transmit data to the processor P_i. Processor P_i calculates the values of initial distribution of temperature in the i-layer according to the formula (2).

5 Conclusion

In the present article The Fourier transform method with dividing points provides the parallel computing for the solution of the direct and inverse problem of the temperature field structure in semi-infinite piece-homogeneous bar Bessel and Weber transforms solve the problems of mathematical physics with spherical symmetry. In the future we will develop the parallel computing algorithms Bessel and Weber transforms.

References

1. *Lebedev N.N., Skalskaya I.P., Uflyand Y.S.* Worked Problems in Applied Mathematics. New York: Dover, 1979.

2. *Il'in V.A.* Convergence of eigenfunction expansions at points of discontinuity of the coefficients of a differential operator//Mathematical Notes. 1977. V. 22. N. 5. P. 870-882.

3. *Yaremko O.E.* Matrix integral Fourier transforms for problems with discontinuous coefficients and transformation operators//Doklady Mathematics. MAIK Nauka. 2007. V. 76. N. 3. P.323-325.

4. Lenyuk M.P. Mathematical modeling of mass transfer in symmetric heterogeneous and nanoporous media with a system of n-interface interactions// Cybernetics and Systems Analysis. 2007. V. 43. P. 94 - 111.

5. *Vasin V.V.* Regularization and iterative approximation for linear ill-posed problems in the space of functions of bounded variation// Tr. Inst. Mat. Mekh. 2002. V. 8. N. 1. P. 189-202.

6. *Tikhonov A.N., Arsenin V.Y.* Solutions of Ill-Posed Problems. Winston; New York, 1977.

7. *Podstrigach Ya.S., Kolyano Yu.M.* Obobshchennaya termomehanika. Kiev: Naukova dumka, 1976.

8. *Brigham E.O.* The Fast Fourier Transform. New York: Prentice-Hall, 2002.

9. *Beck J.V., Blackwell B.St., Clair C.R.* Inverse Heat Conduction. Ill-Posed Problems. New York: J. Wiley, 1985.

Accepted for publication 7.06.2010.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ ДЛЯ ПРЕОБРАЗОВАНИЯ ФУРЬЕ С РАЗРЫВНЫМИ КОЭФФИЦИЕНТАМИ

© **Олег Эммануилович Яремко**
Пензенский государственный педагогический университет им. В.Г. Белинского, Лермонтова 37, Пенза, 440026, Россия, кандидат физико-математических наук, профессор, зав. кафедрой математического анализа, e-mail: yaremki@yandex.ru

© **Наталия Николаевна Яремко**
Пензенский государственный педагогический университет им. В.Г. Белинского, Лермонтова 37, Пенза, 440026, Россия, кандидат физико-математических наук, доцент кафедры математического анализа, e-mail: yaremki@yandex.ru

Ключевые слова: преобразования Фурье с разрывными коэффициентами; прямая и обратная задачи Коши; уравнение теплопроводности.

Исследован параллельный вычислительный алгоритм для решения прямой и обратной задачи о структуре температурного поля в полубесконечном кусочно-однородном стержне. Метод преобразования Фурье с точками деления обеспечивает возможность параллельных вычислений для решения указанных проблем.