Math-Net.Ru

Общероссийский математический портал

А. Я. Дорофеев, Решение систем линейных уравнений при вычислении логарифмов в конечном простом поле, *Матем. вопр. криптогр.*, 2012, том 3, выпуск 1, 5–51

DOI: <https://doi.org/10.4213/mvk47>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением

<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 3.80.181.102

26 апреля 2023 г., 13:14:41

Решение систем линейных уравнений при вычислении логарифмов в конечном простом поле

А. Я. Дорофеев

ФГУП НИИ «Квант», Москва

Получено 05.V.2010

Проведено эмпирическое исследование сложности алгоритмов решения разреженных систем линейных уравнений, которые использовались при вычислении логарифмов в конечных простых полях $GF(p)$ для $p < 10^{135}$.

Ключевые слова: дискретные логарифмы, решето числового поля, разреженные системы линейных уравнений, структурированное гауссово исключение, алгоритм Ланцоша, параллельные вычисления

Solving systems of linear equations arising in the computation of logarithms in a finite prime field

A. Ya. Dorofeev

SUE RDI "Kvant", Moscow

Abstract. Empirical investigations of the computational complexity of algorithms for solving sparse linear systems was conducted for systems appeared in the computation of discrete logarithms in finite prime fields $GF(p)$, $p < 10^{135}$.

Key words: discrete logarithms, number field sieve, sparse linear systems, structured Gaussian elimination, Lanczos algorithm, parallel computations

Citation: *Mathematical Aspects of Cryptography*, 2012, vol. 3, no. 1, pp. 5–51 (Russian).

© 2012 А. Я. Дорофеев

§ 1. Введение

Пусть p — простое число, $GF(p)$ — конечное поле, содержащее p элементов, a — образующий элемент циклической группы $GF(p)^*$. Вычисление такого x , что для заданных p , a и b выполняется $b \equiv a^x \pmod{p}$, называется дискретным логарифмированием в поле $GF(p)$. Стойкость алгоритма открытого распределения ключей по Диффи–Хэллману [1], а также ряда других криптографических алгоритмов, основанных на вычислении показательной функции в конечном поле, определяется вычислительной сложностью дискретного логарифмирования [2]. В 1991 году ЛаМаччия и Одлыжко продемонстрировали слабость одного из указанных алгоритмов, вычислив значение логарифма по простому модулю размера 58 десятичных знаков [3]. Вслед за публикацией [3] появилась серия работ, посвященных практическому вычислению логарифмов в $GF(p)$ для рекордно больших значений характеристики поля p [3–11].

Алгоритмы логарифмирования в $GF(p)$ распадаются на несколько этапов, из которых наибольшую трудоемкость имеют этап формирования разреженной системы линейных уравнений методом перебора (просеивания) и этап решения этой системы.

Этап формирования линейной системы распараллеливается на большое число независимых ветвей, каждая из которых имеет малую вычислительную сложность, поэтому его можно выполнить в распределенной вычислительной среде, коммутируемой, например, по Internet. При этом производительность распределенной вычислительной системы приблизительно равна суммарной производительности входящих в нее вычислительных узлов.

Этап решения линейной системы является «узким местом» алгоритма логарифмирования, которое определяет максимальное значение характеристики поля p , для которой логарифм можно вычислить за приемлемое время [3]. Это связано с тем, что при реализации известных алгоритмов решения разреженных систем (алгоритмов Ланцоша [12], Видемана [13], сопряженных градиентов [14], а также других алгоритмов, основанных на методе пространств Крылова [15]) необходимо итерационно выполнять процедуру умножения разреженной матрицы большой размерности на вектор. При распараллеливании указанной процедуры данные, которые обрабатываются в параллельных процессорах, взаимодействуют между собой в основном цикле алгоритма, что приводит к необходимости использовать дорогостоящую многопроцессорную вычислительную технику, оснащенную средствами высокоскоростной коммутации данных. При этом производительность вычислительной системы существенно снижается за счет затрат времени на синхронизацию ветвей алгоритма и передачу данных между процессорами.

Время решения системы можно сократить, если предварительно исключить из нее методом Гаусса неизвестные, соответствующие наиболее разреженной части матрицы системы [16]. Неизвестные следует исключать так, чтобы минимизировать число ненулевых элементов, добавленных в матрицу системы, — этот метод получил название структурированного гауссова исключения [14]. Вебер показал [4], что время решения ряда систем можно дополнительно сократить, если исключить неизвестные, соответствующие наиболее плотно заполненной части матрицы.

Таблица 1. Рекорды логарифмирования в $GF(p)$

1991, [3]	58	COS	$288\,017 \times 96\,321$	$7\,262 \times 6\,006$	—	L
1996, [4]	85	COS	$119\,951 \times 69\,984$	$51\,855 \times 51\,855$	—	L
1998, [6]	90	COS	$976\,062 \times 674\,564$	$61\,136 \times 61\,036$	92.972	L
1999, [6]	100	NFS	$2\,592\,659 \times 816\,761$	$172\,049 \times 171\,061$	126.404	L
2001, [7]	110	NFS	$1\,437\,324 \times 767\,381$	$160\,329 \times 160\,224$	84.498	L
2001, [8]	120	NFS	$2\,685\,597 \times 1\,242\,551$	$271\,654 \times 271\,552$	83.528	L
2005, [9]	130	NFS	$1\,946\,916 \times 1\,466\,090$	$433\,181 \times 432\,172$	81.545	L
2006, [10]	135	NFS	$2\,739\,345 \times 2\,541\,033$	$565\,000 \times 564\,000$	572.015	L
2007, [11]	160	NFS	—	$2\,177\,226 \times 2\,177\,026$	133.186	W

Результаты логарифмирования в $GF(p)$ для рекордных значений характеристики p хронологически представлены в таблице 1. Во второй колонке таблицы указан размер характеристики поля p в десятичных знаках. В третьей колонке указан алгоритм формирования линейной системы: алгоритм Копперсмита–Одлыжко–Шреппеля (COS) [17] или алгоритм решета числового поля (NFS) [18]. В четвертой колонке указан размер линейной системы, полученной в результате просеивания. В пятой колонке указан размер усеченной системы после исключения неизвестных, в шестой колонке указано среднее число ненулевых элементов в столбце усеченной системы. В седьмой колонке указан итерационный алгоритм, который использовался для решения усеченной системы: алгоритм Ланцоша (L) или алгоритм Видемана (W).

Трудоёмкость формирования линейной системы составляет

$$L(p, v, c) = \exp\{(\ln p)^v (\ln \ln p)^{1-v} (c + o(1))\}$$

двоичных операций, где $\ln p$ — натуральный логарифм p и $p \rightarrow \infty$. Значения констант составляют: $v = 1/2$, $c = 1$ для алгоритма Копперсмита–Одлыжко–Шреппеля [17], $v = 1/3$, $c = (92 + 26\sqrt{13})^{1/3}/3 \approx 1.902$ для алгоритма решета числового поля [18]. Трудоёмкость решения линейной системы асимптотически не превосходит трудоёмкости ее формирования, при этом на прак-

тике параметры алгоритма логарифмирования выбираются так, чтобы время формирования системы и время ее решения приблизительно совпадали.

В статье исследована практическая реализация решения разреженных систем линейных уравнений, полученных при логарифмировании по модулю больших простых чисел. В разделе 2 исследуются особенности линейных систем, полученных методом решета числового поля. В разделе 3 исследована реализация алгоритма Ланцоша, получены оценки сложности для последовательной модели вычислений, измерены времена выполнения отдельных шагов алгоритма. В разделе 4 исследуются методы исключения неизвестных — структурированное гауссово исключение и метод Вебера. Найдены условия, при которых исключение неизвестных приводит к сокращению времени решения системы методом Ланцоша, предложена версия алгоритма гауссова исключения, которая минимизирует время решения системы методом Ланцоша. В разделе 5 исследуются методы распараллеливания алгоритма Ланцоша, получены оценки сложности для параллельной модели вычислений. Предложена версия распараллеленного алгоритма Ланцоша, в которой передача данных между процессорами выполняется во время вычислений.

Представленные в работе алгоритмы были реализованы автором программно, апробированы для модулей логарифмирования размера до 100 десятичных знаков и применены при логарифмировании по модулю размера 135 десятичных знаков [10].

§ 2. Структура линейной системы

Схема решения показательного уравнения

$$a^x = b \quad (1)$$

в простом поле $GF(p)$ методом решета числового поля состоит в следующем [19, раздел 5.5].

1. Выбираем два унитарных многочлена $g_1(x), g_2(x) \in \mathbb{Z}[x]$ степеней n_1, n_2 соответственно, которые неприводимы над кольцом целых чисел $\mathbb{Z}$ и имеют общий корень μ по $\text{mod } p$. Для $j = 1, 2$ обозначаем: α_j — корень $g_j(x)$ в поле комплексных чисел; $\mathbb{K}_j = \mathbb{Q}(\alpha_j)$ — числовое поле, полученное в результате расширения поля рациональных чисел $\mathbb{Q}$ корнем α_j ; $\mathcal{O}_j$ — кольцо целых алгебраических чисел поля $\mathbb{K}_j$.

2. Выбираем факторные базы

$$F_j = \{\mathfrak{p} \mid \mathfrak{p} \text{ — простой идеал кольца } \mathcal{O}_j, \text{Norm}(\mathfrak{p}) < B_j\}, \quad j = 1, 2,$$

где границы гладкости B_j являются параметрами алгоритма.

3. Методом просеивания (решета) находим множество пар $C = \{(c, d)\} \in \mathbb{Z} \times \mathbb{Z}$, для которых идеал $(c + d\alpha_j) \mathcal{O}_j$ разлагается в произведение идеалов из F_j (является гладким относительно F_j) для $j = 1, 2$:

$$(c + d\alpha_j) \mathcal{O}_j = \prod_{\mathfrak{p} \in F_j} \mathfrak{p}^{n(\mathfrak{p}, c, d)}.$$

В результате просеивания строим разреженную целочисленную матрицу A , столбцы которой составлены из пар векторов

$$(n(\mathfrak{p}, c, d) \mid \mathfrak{p} \in F_1), (n(\mathfrak{p}, c, d) \mid \mathfrak{p} \in F_2)$$

для всех $(c, d) \in C$. Множество C должно быть достаточно большим, чтобы матрица A имела ранг $|F_1| + |F_2|$.

4. Пусть $Q = \{q \mid q - \text{простой делитель числа } (p-1), q > B_3, q^2 \nmid (p-1)\}$, где граница B_3 является параметром алгоритма. Для каждого простого $q \in Q$ выполняем:

4.1. В поле $\mathbb{Z}/q\mathbb{Z}$ решаем линейную систему

$$AX = B, \quad (2)$$

где $B = B(a, b)$ — разреженная целочисленная матрица размера $(|F_1| + |F_2|) \times s$, s — небольшое число (параметр алгоритма) в диапазоне $n_1 + n_2 + 1 \leq s \ll \ll |F_1| + |F_2|$.

4.2. По матрице решений X вычисляем такие γ_j , что $\gamma_j = \delta_j^q$ для некоторых $\delta_j \in \mathcal{O}_j, j = 1, 2$. С помощью гомоморфизмов колец

$$\phi_j: \mathbb{Z}[\alpha_j] \rightarrow \mathbb{Z}/p\mathbb{Z}, \quad \phi_j(\alpha_j) = \mu, \quad j = 1, 2,$$

переходим от q -х степеней в кольцах $\mathcal{O}_j, j = 1, 2$, к q -м степеням в поле $\mathbb{Z}/p\mathbb{Z}$. Учитывая, что $\phi_1(c + d\alpha_1) = \phi_2(c + d\alpha_2) = c + d\mu$ для всех $(c, d) \in C$, вычисляем такие $k, l \in \mathbb{Z}$, что $a^k b^l \equiv d^q \pmod{p}$ и $q \nmid l$, тогда

$$x \equiv -\frac{k}{l} \pmod{q}, \quad (3)$$

где x — решение показательного уравнения (1) [4, лемма 1.4].

5. Будем предполагать, что все большие простые числа (превосходящие порог B_3) входят в разложение $p-1$ в первой степени, тогда разложение $p-1$ на простые сомножители имеет вид $(p-1) = (\prod_{(q \in Q)} q) (\prod_{(p \leq B_3)} p^{n_p})$. Используя алгоритм Полига–Хеллмана [20], найдем такие $x_p, p \leq B_3$, что

$$x \equiv x_p \pmod{p^{n_p}}, \quad (4)$$

где x — решение уравнения (1). Единственное значение $x, 0 \leq x < p - 1$, для которого выполняются соотношения (3), (4), будет решением уравнения (1) [4, лемма 1.5]. Оно вычисляется по китайской теореме об остатках.

Различные варианты алгоритма решета числового поля вместе с оценками сложности имеются в работах [4, 6, 18, 19, 21, 22]. Практический интерес представляют модули логарифмирования p , которые являются сильными простыми числами, то есть такими, что $p - 1 = 2q$, где q — простое число. Если p — сильное простое число, то линейная система (2) на шаге 4.1 решается один раз по модулю $q = (p - 1)/2$, а на шаге 5 перебором определяется одно из двух возможных значений $x \pmod{2}$. Все рекордные модули логарифмирования из таблицы 1, а также все модули логарифмирования, которые упоминаются далее в статье, являются сильными простыми числами.

Таблица 2. Линейные системы, сформированные методом решета числового поля

	$S^{(100)}$	$S^{(135)}$
n	365 976	2 739 346
m	314 142	2 541 033
$\neq 0$	6 832 588	57 250 340
$\neq 0/n$	18.669	20.899
$\neq 0/mn$	$5.943 \cdot 10^{-5}$	$8.224 \cdot 10^{-6}$
± 1	6 365 244	52 724 736
$\pm 1/n$	17.392	19.247
$\pm 1/\neq 0$	0.931	0.920
s	8	8

В таблице 2 указаны характеристики систем $S^{(100)}$ и $S^{(135)}$, которые требовалось решить на шаге 4.1 при вычислении логарифмов по модулям $p^{(100)}$ и $p^{(135)}$ размера 100 и 135 десятичных знаков. Системы были сформированы Матюхиным и Дыгиным [10] на основе оригинального варианта алгоритма решета числового поля, предложенного Матюхиным в [18]. Строки n и m таблицы 2 указывают размерность расширенной матрицы АВ системы (2): $n = |C| + s$ — число столбцов в матрице, $m = |F_1| + |F_2|$ — число строк в матрице; строки, обозначенные $\neq 0$ и ± 1 , указывают число ненулевых элементов в матрице системы и число элементов, равных $+1$ или -1 ; строки $\neq 0/n$ и $\pm 1/n$ указывают среднее число ненулевых элементов и элементов, равных ± 1 , в одном столбце матрицы; строка $\neq 0/mn$ указывает отношение числа ненулевых элементов к общему числу элементов матрицы; строка $\pm 1/\neq 0$ указывает отношение числа элементов матрицы, равных ± 1 , к об-

щему числу ненулевых элементов; в строке s указано количество векторов правой части, для которых одновременно решается система (число столбцов в матрице правой части B).

Таблица 3. Распределение значений ненулевых элементов

t	$S^{(100)}$	$S^{(135)}$	t	$S^{(100)}$	$S^{(135)}$
1	6 365 244	52 724 736	12	192	1240
2	290 895	3 232 409	13	106	655
3	97 863	722 743	14	49	328
4	41 012	300 567	15	26	169
5	18 928	137 738	16	15	103
6	9 008	65 190	17	10	47
7	4 591	32 639	18	4	25
8	2 400	16 559	19	3	14
9	1 270	8 461	20	2	6
10	661	4 399	21	—	3
11	309	2 308	23	—	1

В таблице 3 указано распределение значений ненулевых элементов матриц $S^{(100)}$ и $S^{(135)}$. В колонке t указано значение ненулевого элемента, в колонках $S^{(100)}$ и $S^{(135)}$ указано число ненулевых элементов соответствующей матрицы, имеющих это значение.

Таблица 4. Распределение весов столбцов

w	$S^{(100)}$	$S^{(135)}$	w	$S^{(100)}$	$S^{(135)}$
2	2	2	20	65 459	567 969
11	1	—	21	28 594	646 958
12	1	—	22	7 785	521 445
13	50	2	23	1 349	296 389
14	600	29	24	160	118 799
15	4 289	677	25	10	33 145
16	19 365	6 854	26	—	6 388
17	52 627	41 348	27	—	807
18	89 360	147 631	28	—	79
19	96 324	350 821	29	—	3

В таблице 4 указано распределение весов столбцов матриц $S^{(100)}$ и $S^{(135)}$. В колонке w указано значение веса (количества ненулевых эле-

ментов в столбце), в колонках $S^{(100)}$ и $S^{(135)}$ указано число столбцов соответствующей матрицы, обладающих этим весом.

Таблица 5. Распределение ненулевых элементов матрицы по столбцам

Δ	$S^{(100)}$			$S^{(135)}$		
	i	W_c	W_c/i	i	W_c	W_c/i
0.1	36 598	660 521	18.048	273 935	5 599 971	20.442
0.2	73 196	1 332 203	18.200	547 870	11 258 845	20.550
0.3	109 793	2 008 505	18.293	821 804	16 949 663	20.624
0.4	146 391	2 688 923	18.368	1 095 739	22 661 409	20.681
0.5	182 988	3 372 437	18.429	1 369 673	28 391 236	20.728
0.6	219 586	4 058 704	18.483	1 643 608	34 137 622	20.769
0.7	256 184	4 747 180	18.530	1 917 543	39 898 986	20.807
0.8	292 781	5 438 625	18.575	2 191 477	45 670 452	20.840
0.9	329 379	6 133 783	18.622	2 465 412	51 455 963	20.871
1.0	365 976	6 832 588	18.669	2 739 346	57 250 340	20.899

В таблице 5 указано распределение ненулевых элементов по столбцам матриц $S^{(100)}$ и $S^{(135)}$. В колонке i указано число $i = \lceil n\Delta \rceil$, где n — число столбцов матрицы, в колонке W_c указано количество ненулевых элементов в первых i столбцах матрицы. В колонке W_c/i указано среднее значение количества ненулевых элементов в столбце, вычисленное по первым i столбцам матрицы.

Таблица 6. Распределение весов строк

$S^{(100)}$		$S^{(135)}$	
w	r	w	r
365 974	1	2 739 344	1
132 275	1	1 099 993	2
130 894	1	912 478	1
111 959	2	819 325	2
86 133	1	651 374	1
6	24 340	6	185 950
5	33 394	5	259 084
4	43 303	4	351 939
3	49 988	3	438 026
2	47 879	2	454 257

В таблице 6 указано распределение весов строк в матрицах $S^{(100)}$ и $S^{(135)}$ для пяти максимальных и минимальных значений весов. В колонке w указано значение веса, в колонке r указано число строк, обладающих этим весом.

График 1. Распределение ненулевых элементов матрицы $S^{(135)}$ по столбцам (W_c) и строкам (W_r)

В таблице 7 указано распределение ненулевых элементов по строкам матриц $S^{(100)}$ и $S^{(135)}$. В колонке i указано число $i = \lceil m\Delta \rceil$, где m — число строк в матрице, в колонке W_r указано количество ненулевых элементов в первых i строках матрицы. В колонке W_r/i указано среднее значение количества ненулевых элементов в строке, вычисленное по первым i строкам матрицы. В колонке $W_r/\neq 0$ указано, какая часть от общего числа ненулевых элементов находится в первых i строках матрицы. Строки матриц отсортированы по весу (первая строка имеет максимальный вес).

На графике 1 каждая из величин W_c и W_r представлена двумя ветвями. Нижняя ветвь графика W_c (W_r) соответствует расположению столбцов (строк) в матрице, полученной в результате работы алгоритма решета числового поля. Верхняя ветвь графика W_c (W_r) соответствует расположению столбцов (строк) матрицы, отсортированных по весу. Нижняя ветвь графика W_c соответствует таблице 5, верхняя ветвь графика W_r соответствует таблице 7. На нижней ветви графика W_r видны области, соответствующие фак-

Таблица 7. Распределение ненулевых элементов матрицы по строкам

Δ	$S^{(100)}$				$S^{(135)}$			
	i	W_r	W_r/i	$W_r/\neq 0$	i	W_r	W_r/i	$W_r/\neq 0$
10^{-5}	4	741 102	185275.5	0.108	26	14 728 052	566463.538	0.257
10^{-4}	32	1 960 720	61272.5	0.286	255	21 271 183	83416.403	0.371
10^{-3}	315	2 876 020	9130.222	0.420	2 542	28 013 316	11020.187	0.489
10^{-2}	3142	3 968 806	1263.146	0.580	25 411	35 737 897	1406.394	0.624
0.1	31 415	5 219 488	166.146	0.763	254 104	45 104 088	177.502	0.787
0.2	62 829	5 666 588	90.190	0.829	508 207	48 454 624	95.344	0.846
0.3	94 241	5 950 846	63.144	0.870	762 310	50 582 603	66.354	0.883
0.4	125 657	6 163 160	49.047	0.902	1 016 414	52 196 694	51.353	0.911
0.5	157 071	6 334 150	40.326	0.927	1 270 517	53 488 522	42.099	0.934
0.6	188 486	6 475 710	34.356	0.947	1 524 620	54 531 228	35.767	0.952
0.7	219 900	6 597 741	30.003	0.965	1 778 724	55 417 670	31.155	0.967
0.8	251 314	6 691 983	26.627	0.979	2 032 827	56 179 979	27.636	0.981
0.9	282 728	6 769 760	23.944	0.990	2 286 930	56 742 134	24.811	0.991
1.0	314 142	6 832 588	21.749	1.000	2 541 033	57 250 340	22.530	1.000

торным базам F_1 и F_2 (строки матрицы, соответствующие простым идеалам факторных баз, расположены в порядке возрастания норм идеалов).

Таблицы 2–7 и график 1 иллюстрируют основные особенности матриц линейных систем, полученных в результате просеивания методом решета числового поля, а именно:

- ненулевые элементы матриц (степени, с которыми простые идеалы входят в соответствующие разложения) принимают небольшое количество различных значений. Почти все ненулевые элементы матриц равны +1. Остальные ненулевые элементы являются малыми натуральными числами, каждое из которых целиком умещается в один байт оперативной памяти;
- ненулевые элементы матриц распределены неравномерно по строкам и столбцам. Все столбцы являются разреженными, они содержат приблизительно одинаковое число ненулевых элементов (число простых идеалов, на которые разлагаются гладкие идеалы). Несколько строк с максимальным весом (соответствующие идеалам с минимальными нормами) являются плотно заполненными и содержат основную массу ненулевых элементов матриц. Строки матрицы, соответствующие идеалам с максимальными нормами, являются существенно более разреженными, чем столбцы.

§ 3. Алгоритм Ланцоша

Пусть в произвольном поле K требуется найти вектор x , который является решением системы

$$Ax = b, \quad (5)$$

где A — симметричная матрица размера $n \times n$, b — ненулевой вектор размерности n . Итерационный алгоритм решения этой задачи в поле действительных чисел, предложенный Ланцошем [12], основан на использовании пространств Крылова [23, стр. 167] и состоит в следующем.

Алгоритм 1 (алгоритм Ланцоша).

1. Вычислить

$$w_0 = b, \quad v_1 = Aw_0, \quad w_1 = v_1 - w_0 \frac{(v_1, v_1)}{(w_0, v_1)}, \quad x_0 = w_0 \frac{(w_0, b)}{(w_0, v_1)}. \quad (6)$$

2. При $i \geq 1$ в цикле выполнить

$$v_{i+1} = Aw_i, \quad w_{i+1} = v_{i+1} - w_i \frac{(v_{i+1}, v_{i+1})}{(w_i, v_{i+1})} - w_{i-1} \frac{(w_i, v_{i+1})}{(w_{i-1}, v_i)}, \quad (7)$$

$$x_i = x_{i-1} + w_i \frac{(w_i, b)}{(w_i, v_{i+1})}. \quad (8)$$

3. Алгоритм заканчивает работу, как только $(w_k, Aw_k) = 0$ для некоторого $k \leq n$. Если $w_k = 0$, то система имеет решение

$$x = x_{k-1} = \sum_{i=0}^{k-1} \frac{(w_i, b)}{(w_i, Aw_i)} w_i,$$

если $w_k \neq 0$, то алгоритм не находит решения.

При вычислении матрицы решений $X = (x^{(0)} x^{(1)} \dots x^{(s-1)})$ системы

$$AX = B \quad (9)$$

с симметричной матрицей A и с матрицей правой части $B = (b^{(0)} b^{(1)} \dots b^{(s-1)})$, $b^{(j)} \in \text{span}(A^0 b, A^1 b, \dots, A^{n-1} b)$, $1 \leq j \leq s-1$, $b^{(0)} = b$, $b^{(j)} \neq 0$, следует на шагах (6) и (8) алгоритма Ланцоша вычислять:

$$x_0^{(j)} = w_0 \frac{(w_0, b^{(j)})}{(w_0, v_1)}, \quad x_i^{(j)} = x_{i-1}^{(j)} + w_i \frac{(w_i, b^{(j)})}{(w_i, v_{i+1})}, \quad 0 \leq j \leq s-1.$$

Доказательство того факта, что в произвольном поле $\mathbf{K}$ алгоритм Ланцоша завершится (т. е. на некотором шаге будет получен самоортогональный вектор w_k) и что при $w_k = 0$ будет вычислено решение системы (5) (или решение системы (9) для нескольких векторов правой части), представлено в [24, теорема 4].

Матрица A системы (2), полученной в результате просеивания методом решета числового поля, не является симметричной, и ее размерность равна $m \times n$, $m \leq n$. ЛаМаччия и Одлыжко [14] предложили вместо системы (2) решать систему

$$MX = G, \quad (10)$$

где $M = A^T C A$ — симметричная матрица, $G = A^T C B = (g^{(0)} g^{(1)} \dots g^{(s-1)})$, $g^{(j)} = A^T C b^{(j)}$, $0 \leq j \leq s-1$, C — случайная обратимая диагональная матрица. Система (10) решается методом Ланцоша. Если при решении системы (10) получен самоортогональный вектор $w_k \neq 0$ или решение системы (10) не является решением системы (2), то выбирается другая матрица C и процесс повторяется.

Утверждение 1. Если $t \leq n$ и $t \times n$ -матрица A системы (2) имеет ранг t , то системы (2) и (10) эквивалентны.

Доказательство приведено в [25, стр. 181].

Утверждение 2. Пусть $t \leq n$, $t \times n$ -матрица A системы (2) имеет ранг t , S — конечное подмножество множества ненулевых элементов поля $\mathbf{K}$. Пусть элементы обратимой диагональной матрицы C выбираются случайно, независимо и равновероятно из множества S . Тогда вероятность того, что при решении системы (10) в алгоритме Ланцоша будет получен самоортогональный вектор $w_k \neq 0$, не превосходит $\frac{11n^2 - n}{2|S|}$.

Утверждение представляет собой частный случай [25, теорема 6.2].

При вычислении логарифма по модулю сильного простого числа p линейная система (10) решается по модулю простого числа $(p-1)/2$, в качестве S выбирается множество ненулевых элементов поля $\mathbb{Z}/\frac{p-1}{2}\mathbb{Z}$ и оценка вероятности получить самоортогональный вектор в алгоритме Ланцоша составляет $(11n^2 - n)/(p-3)$. Если p достаточно велико, то этой вероятностью можно пренебречь. В частности, для систем $S^{(100)}$ и $S^{(135)}$ (таблица 2) соответствующие вероятности не превосходят 10^{-86} и 10^{-120} .

Все разреженные системы, которые далее упоминаются в работе, решались методом Ланцоша (алгоритм 1) с использованием техники рандомизации ЛаМачча–Одлыжко. Вычислительный эксперимент показал, что для указанных систем решение рандомизированной системы (10) находилось после первого же выбора случайной диагональной матрицы C , полученное решение было решением исходной системы (2), алгоритм Ланцоша заканчивался после выполнения максимально возможного числа итераций, равного количеству строк (рангу) матрицы A .

Рассмотрим реализацию одной итерации алгоритма Ланцоша для линейной системы (10), которая решается в простом поле $\mathbf{K} = \mathbb{Z}/\frac{p-1}{2}\mathbb{Z}$.

Будем полагать, что для хранения данных в оперативной памяти используются массивы $R_0, R_1, R_2^{(j)}, 0 \leq j \leq s-1, R_3, R_4$, в которых размещаются векторы с элементами из поля $\mathbf{K}$, а также скалярные ячейки r_5, r_6, r_7, r_8 , каждая из которых может содержать произвольный элемент поля $\mathbf{K}$ (скаляр). Элементы поля $\mathbf{K}$ представляются в оперативной памяти целыми числами многократной точности [27, глава 4].

Диагональная матрица C хранится в виде последовательности элементов диагонали.

Схема хранения в памяти разреженной матрицы A представляет собой комбинацию стандартной схемы хранения «по столбцам» [26, стр. 22, схема 1] и схемы хранения, использованной Денни [5, стр. 64]. Матрица A хранится в виде совокупности своих столбцов, каждый столбец хранится в виде следующей последовательности:

- j — номер столбца;
- $m_j^{(\neq 0)}$ — число ненулевых элементов столбца, отличных от $+1$ и -1 ;
- $m_j^{(-1)}$ — число ненулевых элементов столбца, равных -1 ;
- $m_j^{(+1)}$ — число ненулевых элементов столбца, равных $+1$;
- $(k_{ij}^{(\neq 0)}, d_{ij})$, $0 \leq i < m_j^{(\neq 0)}$ — последовательность ненулевых элементов столбца, отличных от $+1$ и -1 , $k_{ij}^{(\neq 0)}$ — номер строки, где расположен ненулевой элемент, d_{ij} — значение ненулевого элемента;
- $(k_{ij}^{(-1)})$, $0 \leq i < m_j^{(-1)}$ — последовательность ненулевых элементов столбца, равных -1 , $k_{ij}^{(-1)}$ — номер строки, где расположен элемент -1 ;
- $(k_{ij}^{(+1)})$, $0 \leq i < m_j^{(+1)}$ — последовательность ненулевых элементов столбца, равных $+1$, $k_{ij}^{(+1)}$ — номер строки, где расположен элемент $+1$.

Каждому ненулевому элементу, равному ± 1 , соответствует одна ячейка памяти, где хранится номер строки, в которой расположен этот элемент. Каждому ненулевому элементу, не равному ± 1 , соответствуют две ячейки памяти: в одной хранится номер строки, в другой — значение ненулевого элемента.

Алгоритм 2 (итерация алгоритма Ланцоша с номером i , $i \geq 1$).

Вход: $R_0 = w_i$, $R_1 = -w_{i-1}/(w_{i-1}, v_i)$, $R_2^{(j)} = x_{i-1}^{(j)}$, $0 \leq j < s$.

Выход: $R_0 = w_{i+1}$, $R_1 = -w_i/(w_i, v_{i+1})$, $R_2^{(j)} = x_i^{(j)}$, $0 \leq j < s$.

1. $R_3 := A \cdot R_0$.
2. $R_3 := C \cdot R_3$.
3. $R_4 := A^T \cdot R_3$.
- /* $R_4 = v_{i+1} = A^T C A w_i$ */
4. $r_5 := (R_4, R_0)$.
- /* $r_5 = (w_i, v_{i+1})$ */
5. $r_6 := (R_4, R_4)$.
- /* $r_6 = (v_{i+1}, v_{i+1})$ */
6. $r_8 := -1/r_5$.
7. $R_4 := R_4 + r_5 \cdot R_1$.
- /* $R_4 = v_{i+1} - w_{i-1}(w_i, v_{i+1})/(w_{i-1}, v_i)$ */
8. $R_1 := r_8 \cdot R_0$.
- /* $R_1 = -w_i/(w_i, v_{i+1})$ */

9. Для $j, 0 \leq j < s$, выполнить

9.1. $r_7 := (R_0, -g^{(j)})$.

/* $r_7 = (w_i, -g^{(j)})$ */

9.2. $R_2^{(j)} := R_2^{(j)} + r_7 \cdot R_1$.

/* $R_2^{(j)} = x_i^{(j)} = x_{i-1}^{(j)} + w_i(w_i, g^{(j)})/(w_i, v_{i+1})$ */

10. $R_0 := R_4 + r_6 \cdot R_1$.

/* $R_0 = w_{i+1} = v_{i+1} - w_i(v_{i+1}, v_{i+1})/(w_i, v_{i+1}) - w_{i-1}(w_i, v_{i+1})/(w_{i-1}, v_i)$ */

Алгоритм 3 (вычисление произведения $R' := A \cdot R$).

Вход: $R = (R[0], R[1], \dots, R[n-1]) = (R[j], 0 \leq j < n)$.

Выход: $R' = (R'[0], R'[1], \dots, R'[m-1]) = (R'[i], 0 \leq i < m) = A \cdot R$.

1. Для $i, 0 \leq i < m$, выполнить

1.1. $R'[i] := 0$.

2. Для $j, 0 \leq j < n$, выполнить

2.1. Для $i, 0 \leq i < m_j^{(\neq 0)}$, выполнить

2.1.1. $R'[k_{ij}^{(\neq 0)}] := R'[k_{ij}^{(\neq 0)}] + R[j] \cdot d_{ij}$.

2.2. Для $i, 0 \leq i < m_j^{(-1)}$, выполнить

2.2.1. $R'[k_{ij}^{(-1)}] := R'[k_{ij}^{(-1)}] - R[j]$.

2.3. Для $i, 0 \leq i < m_j^{(+1)}$, выполнить

2.3.1. $R'[k_{ij}^{(+1)}] := R'[k_{ij}^{(+1)}] + R[j]$.

3. Для $i, 0 \leq i < m$, выполнить

3.1. $R'[i] := R'[i] \left(\text{mod } \frac{p-1}{2} \right)$.

Алгоритм 4 (вычисление произведения $R := A^T \cdot R'$).

Вход: $R' = (R'[0], R'[1], \dots, R'[m-1]) = (R'[i], 0 \leq i < m)$.

Выход: $R = (R[0], R[1], \dots, R[n-1]) = (R[j], 0 \leq j < n) = A^T \cdot R'$.

1. Для $j, 0 \leq j < n$, выполнить

1.1. $R[j] := 0$.

1.2. Для $i, 0 \leq i < m_j^{(\neq 0)}$, выполнить

1.2.1. $R[j] = R[j] + R'[k_{ij}^{(\neq 0)}] \cdot d_{ij}$.

1.3. Для $i, 0 \leq i < m_j^{(-1)}$, выполнить

1.3.1. $R[j] = R[j] - R'[k_{ij}^{(-1)}]$.

1.4. Для $i, 0 \leq i < m_j^{(+1)}$, выполнить

$$1.4.1. R[j] = R[j] + R'[k_{ij}^{(+1)}].$$

$$1.5. R[j] := R[j] \left(\bmod \frac{p-1}{2} \right).$$

Из текста алгоритмов 2, 3, 4 видно, что выбранная схема хранения позволяет умножить вектор как на исходную матрицу A , так и на матрицу A^T . Для представления каждого ненулевого элемента, равного ± 1 (они составляют свыше 0.9 от общего числа ненулевых элементов), требуется лишь одна ячейка, что почти вдвое экономит объем используемой памяти по сравнению со стандартной схемой хранения [26].

Пусть матрица A имеет n столбцов и содержит δ ненулевых элементов, из которых γ элементов отличны от ± 1 . Тогда объем памяти, необходимой для хранения матрицы A , составляет

$$4n + \delta + \gamma \text{ ячеек.} \quad (11)$$

В дополнение к памяти (11), предназначенной для хранения матрицы A , в алгоритме 2 используется память для хранения

$$(2s + 3)n + 2m + 4 \text{ элементов поля } \mathbb{Z} / \frac{p-1}{2} \mathbb{Z}. \quad (12)$$

Пусть $m \leq n$, A — разреженная $m \times n$ -матрица ранга m , содержащая δ ненулевых элементов, B — $m \times s$ -матрица с ненулевыми столбцами. Будем полагать, что время выполнения операций $A \cdot R_0$, $C \cdot R_3$, $A^T \cdot R_3$ на первых трех шагах итерации алгоритма Ланцоша составляет соответственно $\tau_1^{(0)}m + \tau_1^{(1)}\delta$, τ_2m , $\tau_3^{(0)}n + \tau_3^{(1)}\delta$, а времена выполнения остальных шагов составляют соответственно τ_4n , τ_5n , τ_6 , τ_7n , τ_8n , $\tau_{9.1}n$, $\tau_{9.2}n$, $\tau_{10}n$. Пусть время, необходимое для контроля вычислений (чтения/записи из внешней памяти векторов $R_0, R_1, R_2^{(j)}$, $0 \leq j < s$, и контроля их значений), составляет $\tau_{11}n + \tau_{12}ns$ в пересчете на одну итерацию. Обозначим $T_0 = \tau_1^{(1)} + \tau_3^{(1)}$, $T_1 = \tau_1^{(0)} + \tau_2$, $T_2 = \tau_3^{(0)} + \tau_4 + \tau_5 + \tau_7 + \tau_8 + \tau_{10} + \tau_{11}$, $T_3 = \tau_{9.1} + \tau_{9.2} + \tau_{12}$, $T_4 = \tau_6$. Будем полагать, что в результате работы алгоритма Ланцоша формируется решение системы (10) (пренебрегая вероятностью неуспешного завершения алгоритма); тогда оценка трудоемкости алгоритма Ланцоша составляет в худшем случае

$$m(T_0\delta + T_1m + T_2n + T_3ns + T_4). \quad (13)$$

В таблице 8 представлены времена выполнения шагов алгоритма Ланцоша при решении систем $S^{(100)}$ и $S^{(135)}$ на процессоре Opteron-285 / 2,6 ГГц. Измерение времени выполнялось по счетчику тактов процессора (Time Stamp Counter). Для указанных систем выполнялось несколько итераций алгоритма

Ланцоша, измерялось время их выполнения, измерялись времена выполнения шагов алгоритма, вычислялись значения коэффициентов T_0, T_1, T_2, T_3, T_4 , после чего масштабированием по количеству итераций вычислялось $time_L$ — время выполнения алгоритма в целом (время выполнения m итераций алгоритма).

В строках $\tau_1^{(1)}$ и $\tau_3^{(1)}$ указаны времена выполнения операции **скаляр** + **число * скаляр** в пересчете на один ненулевой элемент матрицы при умножении разреженных матриц A и A^T на вектор на шагах 1 и 3 соответственно. Здесь и далее **скаляр** — это элемент поля $\mathbb{Z}/\frac{p-1}{2}\mathbb{Z}$, представленный целым числом многократной точности, **число** — это элемент поля $\mathbb{Z}/\frac{p-1}{2}\mathbb{Z}$, представленный небольшим целым числом, которое целиком уместится в одной ячейке оперативной памяти. В строке τ_2 указано время выполнения операции **число * скаляр** в пересчете на один элемент вектора длины m при умножении диагональной матрицы на вектор на шаге 2. В строках $\tau_4, \tau_5, \tau_{9.1}$ указаны времена выполнения операции **скаляр** + **скаляр * скаляр** в пересчете на один элемент вектора длины n при выполнении скалярного произведения двух векторов на шагах 4, 5 и 9.1. В строке τ_6 указано время обращения **скаляра** в поле $\mathbb{Z}/\frac{p-1}{2}\mathbb{Z}$ методом расширенного алгоритма Евклида. В строках $\tau_7, \tau_{9.2}, \tau_{10}$ указаны времена выполнения операции **скаляр** + **число * скаляр** в пересчете на один элемент вектора длины n при выполнении сложения вектора с другим вектором, умноженным на число, на шагах 7, 9.2 и 10. В строке τ_8 указано время выполнения операции **число * скаляр** в пересчете на один элемент вектора длины n при выполнении умножения вектора на число на шаге 8. В строке T_0 указано суммарное время $\tau_1^{(1)} + \tau_3^{(1)}$ выполнения двух операций **скаляр** + **число * скаляр** в пересчете на один ненулевой элемент матрицы A при умножении разреженных матриц на вектор на шагах 1 и 3.

Времена выполнения шагов алгоритма Ланцоша и значения коэффициентов T_0, T_1, T_2, T_3 указаны в 10^{-9} долях секунды (наносекундах). В строке $time_L$ в часах указано суммарное время решения системы методом Ланцоша.

При реализации алгоритма Ланцоша использовалась мобильная библиотека арифметических операций с целыми числами многократной точности, написанная автором на языке C для процессоров с 64-разрядной архитектурой (библиотека апробировалась для процессора Opteron). Время работы программы можно сократить за счет эффективного использования кэш-памяти при блочном умножении матрицы на вектор [28], [29], а также за счет реализации наиболее нагруженных фрагментов программы на языке ассемблера.

§ 4. Структурированный алгоритм Гаусса и метод Вебера

В работе [16] было указано, что время решения разреженной системы методом Ланцоша можно существенно сократить, если предварительно исключить из нее часть строк и столбцов методом Гаусса.

Пусть требуется решить систему вида (2). Оценка (13) времени решения этой системы методом Ланцоша составляет

$$T(m, n) = m(T_0\delta + T_1m + T_2n + T_3ns + T_4),$$

где m — число строк матрицы, n — число столбцов матрицы, $m \leq n$, δ — число ненулевых элементов матрицы, T_0, T_1, T_2, T_3, T_4 — константы, зависящие от времен выполнения шагов алгоритма Ланцоша (таблица 8).

Таблица 8. Времена выполнения шагов алгоритма Ланцоша при решении систем $S^{(100)}$ и $S^{(135)}$ на процессоре Opteron / 2,6 ГГц

	$S^{(100)}$	$S^{(135)}$
$\tau_1^{(0)}$	828.002	958.320
$\tau_1^{(1)}$	207.595	288.115
τ_2	515.626	597.821
$\tau_3^{(0)}$	535.118	647.298
$\tau_3^{(1)}$	176.445	266.679
τ_4	1562.809	2414.777
τ_5	1562.025	2414.654
τ_6	145828.666	222981.002
τ_7	3549.600	5300.784
τ_8	3373.915	5396.090
$\tau_{9,1}$	1494.377	2337.976
$\tau_{9,2}$	3792.580	5667.886
τ_{10}	4152.539	5938.176
T_0	384.040	554.794
T_1	1343.629	1556.141
T_2	17829.588	26890.588
T_3	5286.957	8005.862
T_4	145828.666	222981.002
$time_L$	2185.959	201042.213

Исследуем условия, при которых исключение столбца и строки матрицы системы (2) методом Гаусса приводит к сокращению времени решения системы. Выберем ведущий элемент матрицы. Складывая ведущий столбец, умноженный на подходящий коэффициент, с остальными столбцами матрицы, исключим методом Гаусса все ненулевые элементы в ведущей строке, после чего вычеркнем из матрицы ведущие столбец и строку. Время решения системы, полученной после гауссова исключения, составляет

$$T(m-1, n-1) = (m-1)(T_0(\delta + \Phi) + T_1(m-1) + T_2(n-1) + T_3(n-1)s + T_4),$$

где Φ — число ненулевых элементов, добавленных в матрицу системы. Величина

$$\Delta T(m, n) = \frac{T(m, n) - T(m-1, n-1)}{(m-1)T_0}$$

имеет вид

$$\Delta T(m, n) = \frac{\delta}{m-1} - \Phi + C(s) - R_0(m, n-m, s),$$

где

$$C(s) = \frac{2(T_1 + T_2 + T_3s)}{T_0},$$

$$R_0(m, n-m, s) = \frac{1 - 2(T_1 + T_2 + T_3s) - (n-m)(T_2 + T_3s) - T_4}{(m-1)T_0}.$$

Будем полагать, что для разреженных столбцов матрицы системы (2) выполнено условие Марковица [30]: вектор, равный покомпонентному произведению любых двух различных столбцов, имеет не более одного ненулевого элемента. Тогда $\Phi = (c-2)(r-2) - 2$, где c — количество ненулевых элементов в ведущем столбце, r — количество ненулевых элементов в ведущей строке [6]:

$$\Phi = (c-2)(r-2) - 2 = \left(\frac{\delta}{n} - 2\right)(r-2) - 2 + R_1\left(\frac{\delta}{n}, r, c\right),$$

где $\frac{\delta}{n}$ — среднее число ненулевых элементов в столбце, $R_1\left(\frac{\delta}{n}, r, c\right) = \left(c - \frac{\delta}{n}\right)(r-2)$, тогда

$$\Delta T(m, n) = \frac{\delta}{m-1} - \left(\frac{\delta}{n} - 2\right)(r-2) + 2 + C(s) - R_0(m, n-m, s) - R_1\left(\frac{\delta}{n}, r, c\right).$$

Если $\Delta T(m, n) > 0$, то в результате гауссова исключения время решения системы (2) методом Ланцоша сокращается, если $\Delta T(m, n) < 0$ — то увеличивается. Выполнив эквивалентные преобразования, получим, что знак величины $-\Delta T(m, n)$ совпадает со знаком величины $r - \Pi + R_2$, где

$$\Pi = 2 + \left(\frac{2(T_1 + T_2 + T_3 s)}{T_0} + \frac{\delta}{m-1} + 2 \right) \left(\frac{\delta}{n} - 2 \right)^{-1}, \quad (14)$$

$$R_2 = \left(R_0(m, n - m, s) + R_1 \left(\frac{\delta}{n}, r, c \right) \right) \left(\frac{\delta}{n} - 2 \right)^{-1}.$$

Зафиксируем значение s . Пусть $m \rightarrow \infty$, $n \rightarrow \infty$, так что $m/n \rightarrow 1$, тогда $R_0 \rightarrow 0$. Если ненулевые элементы равномерно распределены по столбцам матрицы системы, то $R_1 = 0$. Ненулевые элементы матрицы системы (2), полученной методом решета числового поля, распределены по столбцам приблизительно равномерно (таблица 5, график 1), поэтому $R_1 \approx 0$. Пусть $(\delta/n)^{-1} R_1 \rightarrow 0$ при $m \rightarrow \infty$, $n \rightarrow \infty$, тогда $R_2 \rightarrow 0$ и $|R_2| < \varepsilon$ для произвольного $\varepsilon > 0$ при достаточно больших m и n .

Из проведенных рассуждений следует *критерий изменения времени решения системы (2) методом Ланцоша*, а именно, исключение столбца и строки матрицы системы методом Гаусса приводит к:

$$- \text{сокращению времени решения системы, если } r < \Pi - \varepsilon, \quad (15)$$

$$- \text{увеличению времени решения системы, если } r > \Pi + \varepsilon, \quad (16)$$

где r — число ненулевых элементов в ведущей строке, Π — порог, определенный соотношением (14), $\varepsilon > 0$ — малое число, которое является параметром критерия.

Структурированный алгоритм Гаусса представляет собой версию алгоритма Гаусса, в которой минимизируется число ненулевых элементов, добавляемых в матрицу линейной системы при ее приведении к треугольному виду. Исключение выполняется для того, чтобы сократить время решения системы методом Ланцоша. В качестве ведущих элементов используются ± 1 .

Алгоритм 5 (структурированный алгоритм Гаусса).

Для матрицы $A = (a_{ij})$ выполнить следующие действия:

1. Выбрать все строки, для которых число ненулевых элементов r не превосходит $\Pi - \varepsilon$, где Π — порог, определенный соотношением (14), $\varepsilon > 0$ — параметр алгоритма. Объявить эти строки активными.
2. Если множество активных строк пусто, то завершить алгоритм.

3. Для каждого $a_{ij} = \pm 1$ в активных строках вычислить вес Марковица — величину $(r_i - 2)(c_j - 2) - 2$, где r_i — число ненулевых элементов в строке i , c_j — число ненулевых элементов в столбце j .
4. Найти элемент $a_{ij}^* = \pm 1$, для которого достигается минимальное значение веса Марковица. Используя a_{ij}^* как ведущий элемент, выполнить гауссово исключение (сложить ведущий столбец, умноженный на подходящий коэффициент, с остальными столбцами матрицы). Удалить строку i и столбец j .
5. Перейти к шагу 1.

Из соотношений (15), (16) следует, что после выполнения очередной итерации алгоритма 5 время решения сжатой системы методом Ланцоша будет сокращаться. Алгоритм 5 завершает работу на шаге 2, как только сжатая матрица станет настолько плотной, что наиболее разреженная строка будет содержать не менее $\Pi - \varepsilon$ ненулевых элементов. Если продолжить выполнение алгоритма 5, то после нескольких итераций матрица системы заполнится настолько плотно, что все строки будут содержать не менее $\Pi + \varepsilon$ ненулевых элементов. Далее после выполнения каждой итерации алгоритма 5 время решения системы методом Ланцоша будет увеличиваться.

Структурированный алгоритм Гаусса был запрограммирован на языке С. Для организации быстрого доступа к элементам матрицы $A = (a_{ij})$ в порядке увеличения их весов Марковица применялось сбалансированное двоичное AVL-дерево [31, стр.492].

Для получения оценок временной сложности был проведен вычислительный эксперимент. Матрицы разреженных систем $S^{(100)}$ и $S^{(135)}$ (полученные методом решета числового поля) сжимались методом структурированного гауссова исключения (для $\Pi = n$, $\varepsilon = 0$ алгоритм 5 завершался на шаге 2, как только в системе оставалось m строк). Для систем, полученных в результате гауссова исключения, по результатам измерений вычислялись коэффициенты T_0, T_1, T_2, T_3, T_4 и время выполнения алгоритма Ланцоша в целом $time_L$ (см. пояснение к таблице 8). По ходу выполнения алгоритма Гаусса для каждой системы вычислялись значение порога Π по формуле (14), а также значение $\min r$ минимума весов строк матрицы системы.

Характеристики систем, полученных из системы $S^{(100)}$ (задача DL100) и системы $S^{(135)}$ (задача DL135) методом структурированного гауссова исключения, представлены в таблице 9 и на графике 2. В строке $\max |\neq 0|$ указано максимальное значение абсолютных величин ненулевых элементов; в строке $\min r$ указано число ненулевых элементов в наиболее разреженной строке матрицы (минимум весов строк); в строке $time_G$ указано время (в минутах) генерации системы методом структурированного гауссова исключения на одном процессоре Opteron-285/2,6 ГГц; остальные строки таблицы 9

Таблица 9. Линейные системы для задач DL100 и DL135

	$S_{55\,000}^{(100)}$	$S_{65\,000}^{(100)}$	$S_{70\,000}^{(100)}$	$S_{75\,000}^{(100)}$	$S^{(100)}$
n	56 000	66 000	71 000	76 000	365 976
m	55 000	65 000	70 000	75 000	314 142
$\neq 0$	93 682 986	26 995 360	19 345 153	15 013 788	6 832 588
$\neq 0/n$	1672.910	409.020	272.466	197.549	18.669
$\neq 0/mn$	$3.041 \cdot 10^{-2}$	$6.292 \cdot 10^{-3}$	$3.892 \cdot 10^{-3}$	$2.633 \cdot 10^{-3}$	$5.943 \cdot 10^{-5}$
± 1	86 280 564	25 292 212	18 104 564	14 028 263	6 365 244
$\pm 1/n$	1540.724	383.215	254.993	184.582	17.392
$\pm 1/\neq 0$	0.920	0.936	0.935	0.934	0.931
$\max \neq 0 $	137	56	56	42	20
$\min r$	20	20	11	5	2
$time_G$	38.398	7.451	5.119	4.011	-
	$S_{500\,000}^{(135)}$	$S_{564\,000}^{(135)}$	$S_{628\,000}^{(135)}$	$S_{692\,000}^{(135)}$	$S^{(135)}$
n	501 000	565 000	629 000	693 000	2 739 345
m	500 000	564 000	628 000	692 000	2 541 033
$\neq 0$	730 414 017	323 188 542	195 954 354	139 323 765	57 250 340
$\neq 0/n$	1457.912	572.015	311.533	201.044	20.899
$\neq 0/mn$	$2.915 \cdot 10^{-3}$	$1.014 \cdot 10^{-3}$	$4.960 \cdot 10^{-4}$	$2.905 \cdot 10^{-4}$	$8.224 \cdot 10^{-6}$
± 1	695 243 804	307 608 541	185 703 456	131 415 717	52 724 736
$\pm 1/n$	1387.712	544.439	295.236	189.633	19.247
$\pm 1/\neq 0$	0.951	0.951	0.947	0.943	0.920
$\max \neq 0 $	114	85	59	52	23
$\min r$	50	21	10	4	2
$time_G$	364.554	233.296	194.077	175.161	—

имеют тот же смысл, что и строки таблицы 2. Из таблицы 9 и графика 2 видно, что почти все ненулевые элементы матриц, полученных методом структурированного гауссова исключения из матриц $S^{(100)}$ и $S^{(135)}$, равны ± 1 , остальные ненулевые элементы — это небольшие целые числа, которые целиком умещаются в одну 32-разрядную ячейку памяти, ненулевые элементы распределены равномерно по столбцам матриц. Из этого следует, что для хранения матриц, полученных в результате гауссова исключения, можно использовать ту же схему хранения, что и для исходных матриц $S^{(100)}$ и $S^{(135)}$.

В таблице 10 для задач DL100 и DL135 представлены значения коэффициентов T_0, T_1, T_2, T_3, T_4 (в наносекундах), полученные в результате из-

График 2. Масштабированное распределение весов матриц по столбцам ($W_c(x \cdot n)/W_c(n)$) и строкам ($W_r(x \cdot m)/W_r(m)$) для задачи DL135

мерений времен выполнения отдельных шагов алгоритма Ланцоша (а также время выполнения алгоритма в целом) на процессоре Opteron-285 / 2,6 ГГц. В строке T_0 указано суммарное время $\tau_1^{(1)} + \tau_3^{(1)}$ выполнения двух операций **скаляр + число * скаляр** в пересчете на один ненулевой элемент матрицы при умножении матриц на вектор на шагах 1 и 3 алгоритма Ланцоша (см. пояснения к таблице 8). В строке $time_L \pm$ указано (в часах) суммарное время, приходящееся на аддитивные операции **скаляр $\pm$ скаляр** на шагах 1 и 3. Суммарное время решения системы методом Ланцоша $time_L$ указано в часах. В строке $time_L \pm / time_L$ указано отношение соответствующих времен. В строке Π указано значение порога, вычисленное по формуле (14). Строка $\min r$ заимствована из таблицы 9, в ней указано значение минимума весов строк матрицы системы. В строке $time_G / time_L$ указано отношение времени генерации системы методом Гаусса (из исходной системы $S^{(100)}$ или $S^{(135)}$) ко времени решения системы методом Ланцоша.

Из таблицы 10 видно, что коэффициенты T_0, T_1, T_2, T_3, T_4 мало изменяются при изменении размеров систем, что позволяет с большой точностью прогнозировать время алгоритма Ланцоша.

Таблица 10. Времена решения линейных систем методом Ланцоша для задач DL100 и DL135 на процессоре Opteron/2,6 ГГц

	$S_{55\,000}^{(100)}$	$S_{65\,000}^{(100)}$	$S_{70\,000}^{(100)}$	$S_{75\,000}^{(100)}$	$S^{(100)}$
T_0	407.001	391.239	389.683	388.045	384.040
T_1	1445.096	1436.278	1434.550	1432.936	1343.629
T_2	18063.152	18258.221	18215.623	18261.995	17829.588
T_3	5473.720	5423.717	5419.096	5423.750	5286.957
T_4	136391.994	143596.853	144646.314	140003.595	145828.666
$time_L$	636.663	265.848	233.535	221.233	2185.959
$\min r$	20	20	11	5	2
Π	3.206	3.817	4.224	4.696	22.628
$time_G/time_L$	$1.005 \cdot 10^{-3}$	$0.467 \cdot 10^{-3}$	$0.365 \cdot 10^{-3}$	$0.302 \cdot 10^{-3}$	—
$time_L \pm$	532.146	178.140	136.839	113.122	211.879
$time_L \pm / time_L$	0.835	0.670	0.585	0.511	0.096
T_3/T_0	13.448	13.862	13.906	13.977	13.766
$m + n$	111 000	131 000	141 000	151 000	680 118
Δ	$7.511 \cdot 10^5$	$9.141 \cdot 10^5$	$9.867 \cdot 10^5$	$10.617 \cdot 10^5$	$50.378 \cdot 10^5$
$\min(\tau_1^{(1)}, \tau_3^{(1)})$	174.778	169.926	169.823	169.910	176.445
$\max(\tau_1^{(1)}, \tau_3^{(1)})$	232.223	221.312	219.860	218.135	207.595
	$S_{500000}^{(135)}$	$S_{564000}^{(135)}$	$S_{628000}^{(135)}$	$S_{692000}^{(135)}$	$S^{(135)}$
T_0	581.892	568.198	561.232	599.815	554.794
T_1	1657.387	1665.583	1664.303	1658.911	1556.141
T_2	27704.107	27551.478	27560.049	27488.692	26890.588
T_3	8243.903	8217.835	8261.940	8153.529	8005.862
T_4	229826.485	220888.565	223620.204	222780.419	222981.002
$time_L$	65662.919	37174.808	29643.458	28635.252	201042.213
$\min r$	50	21	10	4	2
Π	3.229	3.595	4.111	4.602	20.940
$time_G/time_L$	$0.925 \cdot 10^{-4}$	$1.045 \cdot 10^{-4}$	$1.091 \cdot 10^{-4}$	$1.019 \cdot 10^{-4}$	—
$time_L \pm$	56233.281	27483.702	18262.798	15288.437	20826.989
$time_L \pm / time_L$	0.856	0.739	0.616	0.533	0.103
T_3/T_0	14.167	14.462	14.721	13.593	14.430
$m + n$	1 001 000	1 129 000	1 257 000	1 385 000	5 280 378
Δ	$7.096 \cdot 10^6$	$8.170 \cdot 10^6$	$9.258 \cdot 10^6$	$9.419 \cdot 10^6$	$39.529 \cdot 10^6$
$\min(\tau_1^{(1)}, \tau_3^{(1)})$	253.873	248.768	246.124	268.882	266.679
$\max(\tau_1^{(1)}, \tau_3^{(1)})$	328.019	319.429	315.108	330.933	288.115

По мере уплотнения матрицы и сокращения ее размеров в ходе структурированного гауссова исключения время решения систем методом Ланцоша сокращается, достигает минимума при $\Pi \approx \min r$, после чего начинает возрастать, что в точности соответствует соотношениям (15), (16). Время формирования матрицы системы методом Гаусса пренебрежимо мало по сравнению со временем решения сформированной системы методом Ланцоша. В результате гауссова исключения время решения систем (для последовательной модели вычислений) в задачах DL100 и DL135 сокращается в 9.880 и 7.020 раз соответственно.

Вебер [4, стр. 64–66] предложил метод решения системы (2), при котором из матрицы A удаляется часть строк при одновременном увеличении количества правых частей системы (столбцов матрицы B). Этот метод применяется при решении разреженных систем, содержащих плотно заполненные строки, состоящие из целых чисел многократной точности: такие строки исключаются из системы, при этом сокращается время решения системы методом Ланцоша.

Исследуем, позволяет ли метод Вебера сократить время решения системы (2), если удалить из матрицы системы плотно заполненные строки, состоящие из небольших целых чисел (таблица 6).

Метод Вебера состоит в следующем.

Требуется решить систему (2) вида $AX = B$.

Пусть A имеет вид $A = \begin{pmatrix} A' \\ A'' \end{pmatrix}$, где A — $m \times n$ -матрица, A' — плотная $v \times n$ -матрица, A'' — разреженная $t \times n$ -матрица, $v \ll t$, $t + v = m$, $m \leq n$. Матрицу правой части B будем представлять в виде $B = \begin{pmatrix} B' \\ B'' \end{pmatrix}$, где B — $m \times s$ -матрица, B' — $v \times s$ -матрица, B'' — $t \times s$ -матрица.

Пусть r — малое целое число, которое является параметром алгоритма, $0 \leq r \leq n - m$, $r \ll t$.

Пусть $A'' = (A_0'' A_1'')$, где A_0'' — разреженная $t \times (n - r - v)$ -матрица, A_1'' — разреженная $t \times (r + v)$ -матрица. Будем полагать, что ранг матрицы A_0'' равен t . Воспользовавшись алгоритмом Ланцоша, найдем для разреженной матрицы A_0'' такую $(n - r - v) \times (r + v)$ -матрицу Q и такую $(n - r - v) \times s$ -матрицу F , что

$$A_0''(QF) = (-A_1'' - B'').$$

Построим $n \times (r + v)$ -матрицу $S = \begin{pmatrix} Q \\ I \end{pmatrix}$, где I — единичная $(r + v) \times (r + v)$ -матрица. Построим $n \times s$ -матрицу $T = \begin{pmatrix} F \\ 0 \end{pmatrix}$, где 0 — нулевая $(r + v) \times s$ -матрица. Тогда

$$A''S = (A_0'' A_1'')S = 0, \quad A''T + B'' = A_0''F + B'' = 0. \quad (17)$$

Построим $v \times (r + v)$ -матрицу $L = A'S$ и $v \times s$ -матрицу $H = A'T + B'$. Будем полагать, что параметр r выбран так, что ранг матрицы L равен v . Тогда существует решение Y линейной системы

$$LY = H.$$

Найдем (например, методом Гаусса) это решение. Так как размеры v и $(r + v)$ матрицы L пренебрежимо малы по сравнению с размерами m и n матрицы A , то время нахождения решения Y пренебрежимо мало по сравнению со временем работы алгоритма Ланцоша.

Так как $L = A'S$ и $H = A'T + B'$, то $A'SY = A'T + B'$. Из соотношений (17) следует, что $A''SY = 0 = A''T + B''$, тогда $ASY = AT + B$ и

$$A(SY - T) = B,$$

$$\text{т. е. } X = SY - T = \begin{pmatrix} Q \\ I \end{pmatrix} Y - \begin{pmatrix} F \\ 0 \end{pmatrix} - \text{искомое решение системы } AX = B.$$

Время $T(m, n)$ решения системы (2) методом Ланцоша определено соотношением (13). Воспользовавшись методом Вебера для $v = 1, r = 0$, удалим из матрицы (AB) одну плотно заполненную строку и переведем один столбец из матрицы A в матрицу B . Время решения полученной системы методом Ланцоша составляет

$$T(m-1, n-1) = (m-1)(T_0(\delta - \delta') + T_1(m-1) + T_2(n-1) + T_3(n-1)(s+1) + T_4),$$

где δ — число ненулевых элементов матрицы A , δ' — число ненулевых элементов, которые удаляются из матрицы A при удалении строки и столбца. Очевидно, что всегда $\delta' < m + n$.

Предположим, что время решения системы (2) методом Ланцоша сокращается при удалении строки и столбца, т. е. $T(m-1, n-1) < T(m, n)$, что равносильно соотношению $\Delta < \delta'$, где

$$\Delta = \frac{T_3(mn - (s+1)(m+n-1)) - T_1(2m-1) - T_2(m+n-1) - T_4}{T_0(m-1)} - \frac{\delta}{m-1},$$

тогда выполняется соотношение

$$\Delta < \delta' < m + n. \quad (18)$$

Предположим, что $m \rightarrow +\infty$, $n = m + B$ для некоторой константы B ,

$$\frac{\delta}{nm} \rightarrow 0 \text{ при } m \rightarrow +\infty.$$

Тогда при фиксированном s соотношение (18) примет вид

$$\left(\frac{T_3}{T_0} + \gamma_m\right) \cdot n + C + \theta_m = \Delta < \delta' < m + n,$$

где C — константа, $\gamma_m \rightarrow 0$, $\theta_m \rightarrow 0$ при $m \rightarrow +\infty$, $T_0 = \tau_1^{(1)} + \tau_3^{(1)}$ — время выполнения двух операций **скаляр + число * скаляр** (в пересчете на один ненулевой элемент матрицы A) при умножении разреженных матриц A^T и A на вектор на шагах 1 и 3 алгоритма Ланцоша (см. пояснения к таблице 8). Трудоемкость обеих операций линейно зависит от длины представления **скаляра** числом многократной точности. Наконец, $T_3 = \tau_{9.1} + \tau_{9.2} + \tau_{12} > \tau_{9.1}$, где $\tau_{9.1}$ — время выполнения операции **скаляр + скаляр * скаляр** на шаге 9.1 в пересчете на один элемент вектора длины n . Так как трудоемкость мультипликативной операции **скаляр + скаляр * скаляр** растет (как функция от длины представления **скаляра** числом многократной точности) заведомо быстрее, чем линейно [27, раздел 4.3.3.], то при достаточно больших значениях модуля логарифмирования заведомо $T_3/T_0 > 2$. Тогда из соотношения (18) получаем противоречие

$$(2 + \varepsilon) \cdot n < \left(\frac{T_3}{T_0} + \gamma_m\right) \cdot n + C + \theta_m = \Delta < \delta' < m + n < 2n$$

для некоторого $\varepsilon > 0$ при достаточно большом m .

Это означает, что при сделанных предположениях исключение неизвестных из системы методом Вебера приводит к увеличению времени решения системы методом Ланцоша, если все элементы матрицы системы — небольшие целые числа.

По результатам замеров, выполненных для модулей логарифмирования $p^{(100)}$ и $p^{(135)}$, в таблице 10 указаны значения T_3/T_0 , $m + n$ и Δ , вычисленные для соответствующих систем. Из таблиц видно, что для всех систем $T_3/T_0 > 2$ и $m + n < \Delta$, следовательно, $\delta' < \Delta$. Это означает, что при решении указанных систем методом Ланцоша невозможно сократить время решения за счет удаления из матриц плотно заполненных строк методом Вебера. Этот факт был подтвержден экспериментально [10].

§ 5. Распараллеливание алгоритма Ланцоша

Распараллеливание выполняется для вычислительной модели типа MIMD (Multiple Instruction — Multiple Data). Вычислительная модель содержит P параллельных процессоров, имеющих индивидуальные номера от 0 до $P - 1$. Каждый процессор имеет локальную память и работает под управлением собственного потока команд. Все процессоры связаны между собой

коммутационной сетью. Распараллеливание производится по данным: в каждый процессор загружается одна и та же программа, которая выполняется в каждом процессоре для различных исходных данных.

Эффективность распараллеливания алгоритма Ланцоша (алгоритм 2) определяется эффективностью распараллеливания процедуры умножения разреженной матрицы на вектор. Эта процедура является типовой для задач линейной алгебры, поэтому исследованию методов ее распараллеливания посвящено большое число работ [32–40]. Подробный обзор работ по указанной тематике представлен в [29].

Далее исследуются два варианта распараллеливания алгоритма 2, которые различаются методом умножения разреженной матрицы на вектор. Первый алгоритм был предложен в работе [24], он является модификацией алгоритма, который реализовал Денни [5]. Алгоритм хорошо адаптирован для среды параллельного программирования MPI (Message Passing Interface) [41, 42, 43, раздел 5.2], так как глобальные коммутационные операции, используемые в алгоритме, являются в точности операторами среды MPI. Во втором алгоритме применяется метод, который исследовался в работах [33, 35, 39]. Суть метода состоит в том, чтобы совместить вычислительные и коммутационные операции и (в оптимальном варианте) все коммутации выполнять во время векторно-матричных вычислений. Второй алгоритм является более сложным для программирования и отладки, так как он выполняется асинхронно.

Первый вариант распараллеливания алгоритма Ланцоша состоит в следующем.

Матрица A разрезается на P полос по $n_P = n/P$ столбцов в полосе: $A = (A(*, 0) \ A(*, 1) \ \dots \ A(*, P-1))$. Здесь и далее полагаем, что m и n (число строк и столбцов матрицы A) нацело делятся на P (число параллельных процессоров). Каждая полоса $A(*, k)$ хранится в памяти k -го процессора, $0 \leq k < P$, и представляет собой матрицу размера $m \times n_P$.

Каждый из векторов $g^{(j)}$, $0 \leq j < s$, правой части системы (10) разрезается на P векторов одинаковой длины n_P : $g^{(j)} = (g^{(j)}(0) \ g^{(j)}(1) \ \dots \ g^{(j)}(P-1))$. Каждый вектор $g^{(j)}(k)$, $0 \leq j < s$, $0 \leq k < P$, хранится в памяти k -го процессора.

Каждый из массивов $R_0, R_1, R_2^{(j)}$, $0 \leq j < s$, R_4 (см. алгоритм 2) разрезается на P массивов длины n_P каждый, которые располагаются соответственно в массивах $\bar{R}_0, \bar{R}_1, \bar{R}_2^{(j)}$, $0 \leq j < s$, $\bar{R}_4$, в блоках памяти процессоров с последовательными номерами. Так, например, $R_4(0)$ — первые n_P элементов массива R_4 — располагаются в массиве $\bar{R}_4$ в памяти 0-го процессора, $R_4(1)$ —

следующие n_P элементов массива R_4 — располагаются в массиве $\bar{R}_4$ в памяти 1-го процессора, $\dots$, $R_4(P-1)$ — последние n_P элементов массива R_4 — располагаются в массиве $\bar{R}_4$ в памяти $(P-1)$ -го процессора.

Массив R_3 , диагональ C и скаляры r_5, r_6, r_7, r_8 дублируются в памяти каждого процессора. Копию массива R_3 (скаляра r_j), расположенную в памяти процессора с номером k , будем обозначать как $R_3[k]$ ($r_j[k]$).

Алгоритм 6 (первый вариант распараллеливания алгоритма 2).

Вход: $R_0 = w_i$, $R_1 = -w_{i-1}/(w_{i-1}, v_i)$, $R_2^{(j)} = x_{i-1}^{(j)}$, $0 \leq j < s$.

Выход: $R_0 = w_{i+1}$, $R_1 = -w_i/(w_i, v_{i+1})$, $R_2^{(j)} = x_i^{(j)}$, $0 \leq j < s$.

Номер процессора: k , $0 \leq k < P$.

1.1. $R_3 := A(*, k) \cdot \bar{R}_0$.

1.2. $R_3 := \sum_{(l)} R_3[l]$, $0 \leq l < P$.

/* шаг 1.2 выполняется за m операций ALLREDUCE среды MPI */

/* $R_3 = A \cdot R_0$ */

2. $R_3 := C \cdot R_3$.

/* шаг 2 дублирует шаг 2 алгоритма 2 */

3. $\bar{R}_4 := A(*, k)^T \cdot R_3$.

/* $R_4 = v_{i+1} = \bar{A}w_i = A^T C A \cdot R_0$ */

4.1. $r_5 := (\bar{R}_4, \bar{R}_0)$.

4.2. $r_5 := \sum_{(l)} r_5[l]$, $0 \leq l < P$.

/* шаг 4.2 выполняется за одну операцию ALLREDUCE среды MPI */

/* $r_5 = (w_i, v_{i+1}) = (R_4, R_0)$ */

5.1. $r_6 := (\bar{R}_4, \bar{R}_4)$.

5.2. $r_6 := \sum_{(l)} r_6[l]$, $0 \leq l < P$.

/* шаг 5.2 выполняется за одну операцию ALLREDUCE среды MPI */

/* $r_6 = (v_{i+1}, v_{i+1}) = (R_4, R_4)$ */

6. $r_8 := -1/r_5$.

/* шаг 6 дублирует шаг 6 алгоритма 2 */

7. $\bar{R}_4 := \bar{R}_4 + r_5 \cdot \bar{R}_1$.

/* $R_4 = v_{i+1} - w_{i-1}(w_i, v_{i+1})/(w_{i-1}, v_i)$ */

8. $\bar{R}_1 := r_8 \cdot \bar{R}_0$.

/* $R_1 = -w_i/(w_i, v_{i+1}) = r_8 \cdot R_0$ */

9. Для j , $0 \leq j < s$, выполнить

9.1.1. $r_7 := (\bar{R}_0, -g^{(j)}(k))$.

9.1.2. $r_7 := \sum_{(l)} r_7[l]$, $0 \leq l < P$.

/* шаг 9.1.2 выполняется за одну операцию ALLREDUCE среды MPI */

/* $r_7 = (w_i, -g^{(j)}) = (R_0, -g^{(j)})$ */

9.2. $\bar{R}_2^{(j)} := \bar{R}_2^{(j)} + r_7 \cdot \bar{R}_1$.

$$\begin{aligned}
/* R_2^{(i)} &= x_i^{(i)} = x_{i-1}^{(i)} + w_i(w_i, g^{(i)})/(w_i, v_{i+1}) */ \\
10. \bar{R}_0 &:= \bar{R}_4 + r_6 \cdot \bar{R}_1. \\
/* R_0 &= w_{i+1} = v_{i+1} - w_i(v_{i+1}, v_{i+1})/(w_i, v_{i+1}) - w_{i-1}(w_i, v_{i+1})/(w_{i-1}, v_i) */ \\
/* R_0 &= R_4 + r_6 \cdot R_1 */
\end{aligned}$$

За одну операцию ALLREDUCE выбирается сечение памяти (вектор длины P , элементы которого расположены по одному в блоках памяти P различных процессоров), суммируются все элементы сечения и полученная сумма записывается в каждый элемент сечения [42, раздел 6.10.4]. Будем полагать, что одна операция ALLREDUCE выполняется за $\log_2 P$ операций **сдвиг/сложение** над одним сечением памяти. Пусть время операции **сдвиг/сложение** в пересчете на одно сечение памяти составляет на шагах 1.2, 4.2, 5.2, 9.1.2 соответственно $\tau'_{1.2}, \tau'_{4.2}, \tau'_{5.2}, \tau'_{9.1.2}$. Тогда оценка трудоемкости Алгоритма 6 составляет

$$m \left(T_0 \delta_P + (T_1 + T'_1 \log_2 P) m + T_2 \frac{n}{P} + \left(T_3 \frac{n}{P} + T'_3 \log_2 P \right) s + T_4 + T'_4 \log_2 P \right), \quad (19)$$

где коэффициенты T_0, T_1, T_2, T_3, T_4 имеют то же значение, что и в оценке (13) трудоемкости Алгоритма 2, $T'_1 = \tau'_{1.2}$, $T'_3 = \tau'_{9.1.2}$, $T'_4 = \tau'_{4.2} + \tau'_{5.2}$,

$$\delta_P = \max_{0 \leq k < P} \delta(A(*, k)),$$

$\delta(A(*, k))$ — количество ненулевых элементов в матрице $A(*, k)$.

Ненулевые элементы матрицы A , полученной методом решета числового поля, распределены по столбцам приблизительно равномерно (таблица 5, графики 1, 2), поэтому $\delta_P \approx \delta/P$, где δ — количество ненулевых элементов в матрице A . Предполагая, что $\delta_P = \delta/P$, получим оценку трудоемкости алгоритма 6:

$$m \left(T_0 \frac{\delta}{P} + (T_1 + T'_1 \log_2 P) m + T_2 \frac{n}{P} + \left(T_3 \frac{n}{P} + T'_3 \log_2 P \right) s + T_4 + T'_4 \log_2 P \right). \quad (20)$$

Рассмотрим поведение оценки (20) для фиксированных значений δ, m, n, s при росте числа процессоров P . При увеличении числа процессоров до величины

$$P_{min}^{(6)} = \frac{T_0 \delta + T_2 n + T_3 n s}{T'_1 m + T'_3 s + T'_4} \ln 2 \quad (21)$$

трудоемкость (20) сокращается, достигает при $P = P_{min}^{(6)}$ минимума

$$T_{min}^{(6)} = m \left(T_1 m + T_4 + (T'_1 m + T'_3 s + T'_4) \log_2 \left(P_{min}^{(6)} e \right) \right) \quad (22)$$

и начинает возрастать при дальнейшем увеличении числа процессоров.

Для хранения матрицы A в алгоритме 6 требуется в точности тот же объем памяти (11), что и в алгоритме 2, а именно, $4n + \delta + \gamma$ ячеек. В дополнение к памяти (11), предназначенной для хранения матрицы A , в алгоритме 6 используется память для хранения

$$(2s + 3)n + 2mP + 4P \quad \text{элементов поля} \quad \mathbb{Z} / \frac{p-1}{2} \mathbb{Z}. \quad (23)$$

Второй вариант распараллеливания алгоритма Ланцоша состоит в следующем.

Матрица A разрезается на P^2 подматриц по $n_P = n/P$ столбцов и $m_P = m/P$ строк в каждой: $A = (A(i, j))$, $0 \leq i, j < P$. Для фиксированного значения j все матрицы $A(i, j)$, $0 \leq i < P$, хранятся в памяти j -го процессора.

Массивы $R_0, R_1, R_2^{(j)}, R_4$ и векторы правой части системы (10) $g^{(j)}$, $0 \leq j < s$, размещаются в оперативной памяти точно так же, как при первом варианте распараллеливания. Массив R_3 и диагональ C разрезаются на P частей одинаковой длины m_P и размещаются в P различных блоках памяти в точности, как массивы $R_0, R_1, R_2^{(j)}, R_4$ и векторы $g^{(j)}$. Скаляры r_5, r_6, r_7, r_8 дублируются в памяти каждого процессора, как при первом варианте распараллеливания.

Для совмещенного выполнения коммутационных и вычислительных операций используется буферный массив Q , который, размещается в P различных блоках памяти, как массивы $R_0, R_1, R_2^{(j)}, R_3, R_4$.

Алгоритм 7 (второй вариант распараллеливания алгоритма 2).

Вход: $R_0 = w_i$, $R_1 = -w_{i-1}/(w_{i-1}, v_i)$, $R_2^{(j)} = x_{i-1}^{(j)}$, $0 \leq j < s$.

Выход: $R_0 = w_{i+1}$, $R_1 = -w_i/(w_i, v_{i+1})$, $R_2^{(j)} = x_i^{(j)}$, $0 \leq j < s$.

Номер процессора: k , $0 \leq k < P$.

1.1. $\bar{R}_3 := 0$.

1.2. Для t , $0 \leq t < P$, выполнить

1.2.1. Начать процессы: $\bar{Q} := A((t + k + 1) \bmod P, k) \cdot \bar{R}_0$.

$\bar{R}_3 := \bar{R}_3[(k + 1) \bmod P]$.

1.2.2. Ожидать завершения процессов, инициированных на шаге 1.2.1.

1.2.3. $\bar{R}_3 := \bar{R}_3 + \bar{Q}$.

/* $R_3 = A \cdot R_0$ */

2. $\bar{R}_3 := \bar{C} \cdot \bar{R}_3$.

/* $R_3 = CA \cdot R_0$ */

3.1. $\bar{R}_4 := 0$.

3.2. Для t , $0 \leq t < P$, выполнить

3.2.1. Начать процессы: $\bar{R}_4 := \bar{R}_4 + A((t + k) \bmod P, k)^T \cdot \bar{R}_3$.

$\bar{Q} := \bar{R}_3[(k+1) \bmod P]$.

3.2.2. Ожидать завершения процессов, инициированных на шаге 3.2.1.

3.2.3. $\bar{R}_3 := \bar{Q}$.

/* $R_4 = v_{i+1} = \tilde{A}w_i = A^T C A \cdot R_0$ */

/* Дальнейшие шаги Алгоритма 7 совпадают с шагами Алгоритма 6 */

4.1. $r_5 := (\bar{R}_4, \bar{R}_0)$.

4.2. $r_5 := \sum_{(l)} r_5[l], 0 \leq l < P$.

5.1. $r_6 := (\bar{R}_4, \bar{R}_4)$.

5.2. $r_6 := \sum_{(l)} r_6[l], 0 \leq l < P$.

6. $r_8 := -1/r_5$.

7. $\bar{R}_4 := \bar{R}_4 + r_5 \cdot \bar{R}_1$.

8. $\bar{R}_1 := r_8 \cdot \bar{R}_0$.

9. Для $j, 0 \leq j < s$, выполнить

9.1.1. $r_7 := (\bar{R}_0, -g^{(j)}(k))$.

9.1.2. $r_7 := \sum_{(l)} r_7[l], 0 \leq l < P$.

9.2. $\bar{R}_2^{(j)} := \bar{R}_2^{(j)} + r_7 \cdot \bar{R}_1$.

10. $\bar{R}_0 := \bar{R}_4 + r_6 \cdot \bar{R}_1$.

В Алгоритме 7 используются два типа коммутационных операций.

Первый тип — это групповое суммирование-размножение данных, выбранных из сечения памяти (шаги 4.2, 5.2, 9.1.2). Эти операции в алгоритме 7 (как в алгоритме 6) не совмещаются с вычислениями.

Второй тип — это циклический сдвиг данных, выбранных из сечения памяти, вдоль кольца процессоров на одну позицию в направлении уменьшения номеров процессоров (шаги 1.2.1, 1.2.2 и 3.2.1, 3.2.2). Циклические сдвиги в алгоритме 7 выполняются совместно с вычислениями в асинхронном режиме.

Пусть времена циклических сдвигов векторов на шагах 1.2.1 и 3.2.1 составляют соответственно $\tau_{1.2.1}^{(shift)} m/P$ и $\tau_{3.2.1}^{(shift)} m/P$, времена загрузки констант в векторы на шагах 1.1 и 3.1 составляют соответственно $\tau_{1.1}'' m/P$ и $\tau_{3.1}'' n/P$, время сложения векторов на шаге 1.2.3 составляет $\tau_{1.2.3}'' m/P$, время пересылки векторов на шаге 3.2.3 составляет $\tau_{3.2.3}'' m/P$. Тогда оценка трудоемкости алгоритма 7 составляет

$$m \left(\Theta_1 + \Theta_3 + T_0'' m + (T_1 + T_1'') \frac{m}{P} + (T_2 + T_2'') \frac{n}{P} + \right. \\ \left. + \left(T_3 \frac{n}{P} + T_3' \log_2 P \right) s + T_4 + T_4' \log_2 P \right), \quad (24)$$

где коэффициенты $T_1, T_2, T_3, T_4, T_3', T_4'$ имеют то же значение, что и в оценке (19) трудоемкости алгоритма 6, $T_0'' = \tau_{1.2.3}'' + \tau_{3.2.3}''$, $T_1'' = \tau_{1.1}''$,

$$T_2'' = \tau_{3,1}'',$$

$$\Theta_1 = \sum_{(0 \leq j < P)} \max \left\{ \tau_1^{(1)} \max_{(0 \leq i < P)} \delta(A(i, (i+j) \bmod P)), \tau_{1.2.1}^{(shift)} \frac{m}{P} \right\}, \quad (25)$$

$$\Theta_3 = \sum_{(0 \leq j < P)} \max \left\{ \tau_3^{(1)} \max_{(0 \leq i < P)} \delta(A(i, (i+j) \bmod P)), \tau_{3.2.1}^{(shift)} \frac{m}{P} \right\}, \quad (26)$$

$\delta(A(i, j))$ — количество ненулевых элементов в матрице $A(i, j)$, $\tau_1^{(1)}$ и $\tau_3^{(1)}$ имеют то же значение, что и в оценке (13) трудоемкости алгоритма 2 (таблицы 8, 10).

Исследуем оценку (24). Будем полагать, что ненулевые элементы распределены равномерно по матрицам $A(i, j)$, т. е. $\delta(A(i, j)) = \delta/P^2$, $0 \leq i, j < P$, где δ — число ненулевых элементов матрицы A . Пусть $\tau_{1.2.1}^{(shift)} = \tau_{3.2.1}^{(shift)} = \tau^{(shift)}$. Тогда соотношения (25), (26) примут вид

$$\Theta_1 = \frac{m}{P} \sum_{(0 \leq j < P)} \max \left\{ \frac{1}{P} \tau_1^{(1)} \frac{\delta}{m}, \tau^{(shift)} \right\}, \quad (27)$$

$$\Theta_3 = \frac{m}{P} \sum_{(0 \leq j < P)} \max \left\{ \frac{1}{P} \tau_3^{(1)} \frac{\delta}{m}, \tau^{(shift)} \right\}. \quad (28)$$

Обозначим $\tau_{min}^{(1)} = \min \{ \tau_1^{(1)}, \tau_3^{(1)} \}$, $\tau_{max}^{(1)} = \max \{ \tau_1^{(1)}, \tau_3^{(1)} \}$, тогда из соотношений (27), (28) следует, что

$$\Theta_1 + \Theta_3 = \begin{cases} \frac{T_0 \delta}{P}, & \text{если } \tau^{(shift)} \leq \frac{1}{P} \tau_{min}^{(1)} \frac{\delta}{m}, \\ 2m \tau^{(shift)}, & \text{если } \tau^{(shift)} \geq \frac{1}{P} \tau_{max}^{(1)} \frac{\delta}{m}, \end{cases}$$

где $T_0 = \tau_1^{(1)} + \tau_3^{(1)}$ (см. оценку (13) трудоемкости алгоритма 2).

Пусть $len(\frac{p-1}{2})$ — объем памяти, которая необходима для хранения одного элемента поля $\mathbb{Z}/\frac{p-1}{2}\mathbb{Z}$, $V^{(shift)} = len(\frac{p-1}{2})/\tau^{(shift)}$ — скорость пересылки данных по кольцу при выполнении циклического сдвига на один процессор,

$$V_{max}^{(shift)} = \frac{m \cdot len\left(\frac{p-1}{2}\right)}{\tau_{min}^{(1)} \cdot \delta}, \quad V_{min}^{(shift)} = \frac{m \cdot len\left(\frac{p-1}{2}\right)}{\tau_{max}^{(1)} \cdot \delta}, \quad (29)$$

тогда

$$\Theta_1 + \Theta_3 = \frac{T_0 \delta}{P}, \quad \text{если } \frac{V^{(shift)}}{P} \geq V_{max}^{(shift)}, \quad (30)$$

$$\Theta_1 + \Theta_3 = 2m\tau^{(shift)}, \quad \text{если } \frac{V^{(shift)}}{P} \leq V_{min}^{(shift)}. \quad (31)$$

Зафиксируем значения δ, m, n, s и будем увеличивать число процессоров P .

Если $P \leq V^{(shift)}/V_{max}^{(shift)}$, то все пересылки данных по кольцу выполняются во время арифметических вычислений и оценка (24) трудоемкости алгоритма 7 принимает вид

$$m \left(\frac{T_0 \delta}{P} + T_0'' m + (T_1 + T_1'') \frac{m}{P} + (T_2 + T_2'') \frac{n}{P} + \right. \\ \left. + \left(T_3 \frac{n}{P} + T_3' \log_2 P \right) s + T_4 + T_4' \log_2 P \right). \quad (32)$$

Если $V^{(shift)}/V_{min}^{(shift)} \leq P$, то арифметические вычисления выполняются во время пересылок по кольцу и оценка (24) принимает вид

$$m \left((T_0'' + 2\tau^{(shift)}) m + (T_1 + T_1'') \frac{m}{P} + (T_2 + T_2'') \frac{n}{P} + \right. \\ \left. + \left(T_3 \frac{n}{P} + T_3' \log_2 P \right) s + T_4 + T_4' \log_2 P \right). \quad (33)$$

При увеличении P до величины

$$P_{min}^{(7)} = \frac{(T_1 + T_1'') m + (T_2 + T_2'') n + T_3 n s}{T_3' s + T_4'} \ln 2 \quad (34)$$

трудоемкость (33) сокращается, достигает минимума

$$T_{min}^{(7)} = m \left((T_0'' + 2\tau^{(shift)}) m + T_4 + (T_3' s + T_4') \log_2 \left(P_{min}^{(7)} e \right) \right) \quad (35)$$

при $P = P_{min}^{(7)}$ и возрастает при дальнейшем увеличении P .

На практике при больших значениях m и n значение $P_{min}^{(7)}$ превосходит количество параллельных процессоров вычислительной системы. В этом случае при увеличении числа используемых процессоров P трудоемкость алгоритма (33) сокращается и достигает минимума при P , равном количеству процессоров вычислительной системы.

Для хранения матрицы A в алгоритме 7 требуется $4n + \delta + \gamma$ ячеек памяти — в точности тот же объем памяти (11), что в алгоритмах 2 и 6. В дополнение к памяти (11), предназначенной для хранения матрицы A , в алгоритме 7 используется память для хранения

$$(2s + 3)n + 3m + 4P \quad \text{элементов поля } \mathbb{Z} / \frac{p-1}{2} \mathbb{Z}. \quad (36)$$

В таблице 11 указана (в Мбайт/сек.) скорость пересылки данных по кольцу процессоров $V^{(shift)}$, измеренная для коммутационных сетей Myrinet и InfiniBand при изменении числа процессоров в кольце. Для каждого значения скорости пересылки вычисляется скорость, приведенная к числу процессоров $V^{(shift)}/P$, а также $\tau_{100}^{(shift)}$ и $\tau_{135}^{(shift)}$: времена (в наносекундах) пересылки по кольцу одного элемента поля $\mathbb{Z} / \frac{p^{(100)} - 1}{2} \mathbb{Z}$ и поля $\mathbb{Z} / \frac{p^{(135)} - 1}{2} \mathbb{Z}$ (44 и 56 байт соответственно).

Таблица 11. Скорости и времена пересылки данных по кольцу процессоров

P	Myrinet				InfiniBand			
	$V^{(shift)}$	$\frac{V^{(shift)}}{P}$	$\tau_{100}^{(shift)}$	$\tau_{135}^{(shift)}$	$V^{(shift)}$	$\frac{V^{(shift)}}{P}$	$\tau_{100}^{(shift)}$	$\tau_{135}^{(shift)}$
2	88.631	44.315	473.437	602.556	247.088	123.544	169.824	216.140
4	88.059	22.014	476.514	606.472	224.912	56.228	186.568	237.451
8	86.808	10.851	483.379	615.210	233.917	29.239	179.386	228.310
16	87.148	5.446	481.494	612.810	171.335	10.708	244.909	311.702
20	85.875	4.293	488.636	621.900	225.716	11.285	185.904	236.605
24	88.217	3.675	475.664	605.390	210.635	8.776	199.215	253.546
32	81.761	2.555	513.218	653.186	189.240	5.913	221.737	282.211
40	87.963	2.199	477.033	607.133	191.849	4.796	218.721	278.373
48	88.271	1.838	475.372	605.020	182.571	3.803	229.837	292.520
56	45.048	0.804	931.477	1185.516	157.485	2.812	266.448	339.116
64	43.146	0.674	972.539	1237.777	210.611	3.290	199.236	253.574
72	44.964	0.624	933.209	1187.721	234.061	3.250	179.276	228.169
80	41.365	0.517	1014.419	1291.079	256.212	3.202	163.776	208.443
88	44.834	0.509	935.921	1191.172	237.052	2.693	177.014	225.291
90	45.712	0.507	917.948	1168.298	216.880	2.409	193.478	246.245

В таблице 12 указаны граничные значения скорости пересылки $V_{min}^{(shift)}$ и $V_{max}^{(shift)}$, вычисленные по формулам (29) для линейных сис-

тем, соответствующих модулям логарифмирования $p^{(100)}$ и $p^{(135)}$ (значения $\tau_{min}^{(1)} = \min \{ \tau_1^{(1)}, \tau_3^{(1)} \}, \tau_{max}^{(1)} = \max \{ \tau_1^{(1)}, \tau_3^{(1)} \}$ указаны в таблице 10).

Таблица 12. Граничные значения скорости пересылки данных по кольцу для процессора Opteron / 2,6 ГГц

	$S_{55\,000}^{(100)}$	$S_{65\,000}^{(100)}$	$S_{70\,000}^{(100)}$	$S_{75\,000}^{(100)}$	$S^{(100)}$
$V_{min}^{(shift)}$	0.106	0.456	0.690	0.960	9.293
$V_{max}^{(shift)}$	0.140	0.594	0.894	1.233	10.934
	$S_{500\,000}^{(135)}$	$S_{564\,000}^{(135)}$	$S_{628\,000}^{(135)}$	$S_{692\,000}^{(135)}$	$S^{(135)}$
$V_{min}^{(shift)}$	0.111	0.291	0.543	0.801	8.227
$V_{max}^{(shift)}$	0.144	0.374	0.695	0.986	8.888

Из таблиц 11 и 12 видно, что при малом числе процессоров для всех линейных систем выполняется соотношение (30), т.е. при умножении разреженной матрицы на вектор все пересылки выполняются во время вычислений. При большом числе процессоров соотношение (30) выполняется только для систем, полученных в результате структурированного гауссова исключения. Для исходных систем $S^{(100)}$, $S^{(135)}$, сформированных в результате просеивания, выполнено соотношение (31), то есть все вычисления выполняются во время пересылок данных по кольцу, которые и определяют трудоемкость умножения разреженной матрицы на вектор.

Сравним оценки трудоемкости (22) и (35) алгоритмов 6 и 7. Пусть

$$m \rightarrow +\infty, \quad n = m + C_0, \quad \frac{\delta}{n} \rightarrow C_1,$$

где C_0, C_1 — константы. Положим $T'_1 = T'_3 = T'_4 > \tau^{(shift)}$. За счет увеличения буферного массива Q и усложнения алгоритма асинхронного обмена можно исключить из алгоритма 7 шаги 1.2.3 и 3.2.3, поэтому можно положить $T''_0 = \tau''_{1.2.3} + \tau''_{3.2.3} = 0$. Тогда из соотношений (21), (22), (34), (35) следует, что при фиксированном s (количестве правых частей системы)

$$\frac{P_{min}^{(6)}}{P_{min}^{(7)}} \rightarrow 0, \quad \frac{T_{min}^{(6)}}{T_{min}^{(7)}} > \Phi + \theta_m,$$

$$\text{где } \Phi = \frac{T_1}{2\tau^{(shift)}} + \frac{1}{2} \log_2 \left(\frac{T_0 + T_2 + sT_3}{\tau^{(shift)}} \right) + 0.456, \quad \theta_m \rightarrow 0 \text{ при } m \rightarrow +\infty.$$

При достаточно больших значениях модуля логарифмирования заведомо $T_1/\tau^{(shift)} > 2$ (см. исследование соотношения (18) при анализе метода

Вебера), поэтому для достаточно больших значений m

$$P_{min}^{(6)} \ll P_{min}^{(7)}, \quad T_{min}^{(6)} > T_{min}^{(7)}. \quad (37)$$

Алгоритмы 6 и 7 (А6 и А7) были запрограммированы на языке Си в среде параллельного программирования MPI. Для сравнительного исследования алгоритмов 6 и 7 были выбраны модули логарифмирования $p^{(39)}$, $p^{(49)}$, $p^{(60)}$, $p^{(75)}$, $p^{(80)}$, $p^{(90)}$, содержащие соответственно 39, 49, 60, 75, 80 и 90 десятичных знаков. Для каждого модуля методом решета была сформирована разреженная система уравнений, при этом для модулей $p^{(39)}$ и $p^{(49)}$ был использован метод линейного решета с последующим исключением неизвестных методом Гаусса (SGE) [24], для модулей $p^{(60)}$ и $p^{(90)}$ был использован метод решета числового поля (NFS), а для модулей $p^{(75)}$ и $p^{(80)}$ был использован метод решета в целых числах Гаусса (COS). Генерация систем методом COS и NFS выполнена Матюхиным и Дыгиным [10].

Алгоритмы 6 и 7 выполнялись на многопроцессорной системе, содержащей узлы (2 x Pentium-3 / 1 ГГц / 2 Гбайт RAM), связанные коммутационной сетью Myrinet [44]. Каждая система решалась методом Ланцоша, после чего вычислялись значения логарифмов. В таблице 13 для алгоритмов 6 и 7 указаны (в минутах) времена решения линейных систем при изменении числа используемых параллельных процессоров. Из таблицы 13 видно, что при увеличении числа процессоров времена работы алгоритмов 6 и 7 в точности соответствуют поведению оценок трудоемкости (20), (24), (37). На основе указанных оценок, а также на основе результатов вычислительных экспериментов (таблицы 11, 12, 13) для решения линейных систем, соответствующих рекордным модулям логарифмирования $p^{(100)}$ и $p^{(135)}$, был выбран алгоритм 7.

Значение оценки (24) трудоемкости алгоритма 7 может существенно увеличиться, если ненулевые элементы неравномерно распределены по матрице A .

Пусть данные пересылаются между процессорами настолько быстро, что для всех j в диапазоне $0 \leq j < P$ выполнено

$$\max_{(0 \leq i < P)} \delta(A(i, (i + j) \bmod P)) \geq \frac{\tau^{(shift)} m}{\tau_{min}^{(1)} P}, \quad (38)$$

т. е. все пересылки в алгоритме 7 выполняются во время вычислений и время пересылок не влияет на оценку (24). Тогда в оценке (24) получим $\Theta_1 + \Theta_3 = T_0 \Theta$, где

$$\Theta = \sum_{(0 \leq j < P)} \left(\max_{(0 \leq i < P)} \delta(A(i, (i + j) \bmod P)) \right) \quad (39)$$

Таблица 13. Времена работы алгоритмов 6 и 7

P	$p^{(39)}$		$p^{(49)}$		$p^{(60)}$	
	$A6$	$A7$	$A6$	$A7$	$A6$	$A7$
2	6.325	7.381	146.796	157.912	284.900	309.739
4	3.365	3.958	75.202	95.586	154.553	176.167
8	1.945	2.192	39.874	54.055	89.681	107.135
16	1.542	1.349	24.681	25.003	59.867	70.720
20	1.759	1.172	22.672	21.308	57.426	62.912
24	1.891	1.067	20.912	18.404	55.165	58.451
32	2.139	0.927	19.739	15.588	58.903	46.272
40	3.078	0.836	21.557	13.741	61.495	42.172
48	3.730	0.800	23.302	12.377	72.619	41.630

P	$p^{(75)}$		$p^{(80)}$		$p^{(90)}$	
	$A6$	$A7$	$A6$	$A7$	$A6$	$A7$
2	688.291	819.481	1313.844	1537.135	22264.944	23302.951
4	402.939	523.246	765.232	994.212	11883.386	13136.153
8	265.099	402.183	495.242	691.063	6715.724	7764.170
16	208.724	274.016	384.970	525.851	4235.544	5082.653
20	205.701	251.496	374.375	499.362	3786.636	4489.080
24	201.620	240.766	362.602	456.392	3449.582	4032.715
32	199.202	227.314	355.930	434.416	3052.873	3597.230
40	219.908	216.432	381.242	407.731	2939.527	3131.916
48	251.345	222.780	437.163	444.815	2832.147	2882.854

— трудоемкость процедуры умножения матрицы A на вектор R_0 на шагах 1.2.1, 1.2.2 (матрицы A^T на вектор R_3 на шагах 3.2.1, 3.2.2) алгоритма 7 в пересчете на количество операций **скаляр + число * скаляр**. Эффективность распараллеливания для указанной процедуры имеет вид

$$E(P) = \frac{T(1)}{P T(P)} = \frac{\delta}{P \Theta}, \quad (40)$$

где $\delta = T(1)$ — трудоемкость реализации алгоритма на системе с одним процессором, $\Theta = T(P)$ — трудоемкость реализации алгоритма на системе с P процессорами, P — число параллельных процессоров в системе. Эффективность распараллеливания лежит в диапазоне $1/P \leq E(P) \leq 1$.

Если ненулевые элементы распределены равномерно по матрицам $A(i, j)$, то оценка (39) принимает минимально возможное значение $\Theta = \delta/P$. При этом эффективность распараллеливания (40) имеет максимально возмож-

ное значение $E(P) = 1$, то есть умножение разреженной матрицы на вектор в алгоритме 7 выполняется с максимальным ускорением, равным количеству параллельных процессоров вычислительной системы.

Если для каждого j в диапазоне $0 \leq j < P$ все матрицы $A(i, (i + j) \bmod P)$, $0 \leq i < P$, кроме, может быть, одной, нулевые, то оценка (39) принимает максимально возможное значение $\Theta = \delta$. При этом эффективность распараллеливания (40) имеет минимально возможное значение $E(P) = 1/P$, т.е. в алгоритме 7 умножение разреженной матрицы на вектор выполняется на параллельной вычислительной системе в последовательном режиме без какого-либо ускорения.

В таблице 14 для матриц систем, соответствующих модулям логарифмирования $p^{(100)}$ и $p^{(135)}$, указаны значения эффективности распараллеливания $E(P)$, полученные из соотношений (39), (40). Основная масса ненулевых элементов матриц сосредоточена в небольшом множестве плотных строк (таблица 7, графики 1 и 2), которые неравномерно распределены по разреженным матрицам. Это приводит к тому, что эффективность распараллеливания $E(P)$ для матриц, указанных в таблице 14, близка к минимально возможной.

Для увеличения эффективности распараллеливания (сокращения времени) умножения разреженных матриц на вектор предлагается переставить строки матрицы A так, чтобы во всех подматрицах $A(i, j)$ содержалось приблизительно одинаковое количество ненулевых элементов. Суть эвристического алгоритма перестановки строк состоит в следующем. Матрица A разрезается по строкам на P полос одного размера: по $m_P = m/P$ строк в каждой полосе (полагаем, что m кратно P). Строки матрицы A переставляются так, чтобы в каждую полосу попало приблизительно одинаковое количество ($\approx \delta/P$) ненулевых элементов. Так как ненулевые элементы распределены по столбцам матрицы A приблизительно равномерно (таблица 5, графики 1 и 2), то в каждый блок $A(i, j)$, $0 \leq i < P$, $0 \leq j < P$, попадет приблизительно одинаковое количество ($\approx \delta/P^2$) ненулевых элементов и величина Θ будет близка к минимально возможному значению δ/P .

Алгоритм 8 (жадный алгоритм перестановки строк).

Для матрицы A выполнить:

1. Для каждой строки с номером i , $0 \leq i < m$, вычислить ее вес $w(i)$ (количество ненулевых элементов в строке).
2. Отсортировать номера строк в порядке убывания весов:
 $w(i_0) \geq w(i_1) \geq \dots \geq w(i_k) \geq w(i_{k+1}) \geq \dots \geq w(i_{m-1})$.
3. Установить указатели и счетчики: для t , $0 \leq t < P$, выполнить
 - 3.1. $pointer(t) := t \cdot m_P$.
 - 3.2. $counter(t) := 0$.

Таблица 14. Эффективность умножения матрицы на вектор

	DL100							
	$S_{55\,000}^{(100)}$		$S_{65\,000}^{(100)}$		$S_{75\,000}^{(100)}$		$S^{(100)}$	
P	$E(P)$	$E'(P)$	$E(P)$	$E'(P)$	$E(P)$	$E'(P)$	$E(P)$	$E'(P)$
2	0.820	0.980	0.812	0.977	0.810	0.969	0.768	0.987
4	0.511	0.979	0.481	0.972	0.470	0.961	0.419	0.981
8	0.319	0.973	0.283	0.969	0.271	0.960	0.228	0.978
16	0.204	0.968	0.168	0.958	0.157	0.955	0.124	0.972
20	0.178	0.967	0.143	0.959	0.132	0.952	0.101	0.859
24	0.160	0.958	0.125	0.950	0.115	0.950	0.086	0.725
32	0.136	0.958	0.102	0.954	0.092	0.945	0.067	0.553
40	0.121	0.954	0.087	0.947	0.078	0.938	0.055	0.447
48	0.110	0.947	0.077	0.939	0.068	0.935	0.047	0.375
56	0.102	0.941	0.069	0.936	0.060	0.931	0.041	0.323
64	0.096	0.941	0.064	0.930	0.055	0.927	0.036	0.284
72	0.092	0.934	0.059	0.926	0.050	0.924	0.032	0.253
80	0.088	0.932	0.055	0.920	0.047	0.911	0.030	0.228
88	0.084	0.921	0.052	0.919	0.044	0.913	0.027	0.208
90	0.083	0.929	0.051	0.916	0.043	0.910	0.027	0.203

	DL135							
	$S_{500\,000}^{(135)}$		$S_{564\,000}^{(135)}$		$S_{628\,000}^{(135)}$		$S^{(135)}$	
P	$E(P)$	$E'(P)$	$E(P)$	$E'(P)$	$E(P)$	$E'(P)$	$E(P)$	$E'(P)$
2	0.787	0.980	0.788	0.977	0.788	0.974	0.763	0.991
4	0.459	0.974	0.452	0.970	0.448	0.964	0.413	0.989
8	0.267	0.971	0.258	0.966	0.253	0.959	0.223	0.988
16	0.156	0.971	0.148	0.965	0.143	0.956	0.120	0.986
20	0.132	0.968	0.124	0.964	0.119	0.953	0.098	0.947
24	0.115	0.967	0.107	0.964	0.103	0.955	0.083	0.808
32	0.092	0.969	0.085	0.964	0.082	0.955	0.064	0.617
40	0.078	0.965	0.072	0.961	0.068	0.951	0.053	0.499
48	0.069	0.961	0.062	0.960	0.059	0.952	0.045	0.419
56	0.061	0.962	0.055	0.957	0.052	0.947	0.039	0.361
64	0.056	0.962	0.050	0.956	0.047	0.945	0.034	0.317
72	0.051	0.954	0.045	0.954	0.042	0.946	0.031	0.282
80	0.047	0.958	0.042	0.955	0.039	0.945	0.028	0.255
88	0.044	0.957	0.039	0.949	0.036	0.942	0.026	0.232
90	0.044	0.955	0.038	0.949	0.035	0.940	0.025	0.227

3.3. $weight_counter(t) := 0$.

4. Сформировать перестановку номеров строк (i'_q) , $0 \leq q < m$:

для k , $0 \leq k < m$, выполнить

4.1. Найти t_{min} , для которого $counter(t_{min}) < m_P$ и счетчик $weight_counter$ имеет минимальное значение, $weight_counter(t_{min}) \leq weight_counter(t)$ для $0 \leq t < P$.

4.2. $i'_{pointer(t_{min})} := i_k$.

4.3. $pointer(t_{min}) := pointer(t_{min}) + 1$.

4.4. $counter(t_{min}) := counter(t_{min}) + 1$.

4.5. $weight_counter(t_{min}) := weight_counter(t_{min}) + w(i_k)$.

5. Переставить строки матрицы A : строку с номером i'_q переставить на место строки с номером q , $0 \leq q < m$.

В таблице 14 указаны значения эффективности распараллеливания $E'(P)$, вычисленные для процедуры умножения матрицы на вектор по формулам (39), (40) после перестановки строк соответствующих матриц по алгоритму 8. Из таблицы видно, что для матриц систем, полученных в результате гауссова исключения, эффективность распараллеливания процедуры умножения матрицы на вектор после перестановки строк по алгоритму 8 близка к максимально возможному значению 1.

В таблицах 15–18 для задач DL100 и DL135 указаны (в часах) времена решения линейных систем методом Ланцоша (алгоритм 7) с учетом перераспределения ненулевых элементов матриц (алгоритм 8) в зависимости от числа используемых вычислительных ядер [45]. Времена получены в результате измерений, выполненных на двух вычислительных системах. Первая серия измерений проводилась на многопроцессорной системе, содержащей узлы (2 x Pentium-3 / 1 ГГц / 2 Гбайт RAM), связанные коммутационной сетью Myrinet. Вторая серия измерений проводилась на многопроцессорной системе, содержащей узлы (2 x Opteron-285 / 2,6 ГГц / 16 Гбайт RAM), связанные коммутационной сетью InfiniBand. В сводной таблице 19 указаны времена решения линейных систем $T(P)$, а также соответствующие им относительные оценки эффективности распараллеливания $E(P)$. Из таблиц 9 и 15–19 видно, что для рекордных модулей логарифмирования $p^{(100)}$ и $p^{(135)}$ времена сжатия матриц систем методом Гаусса пренебрежимо малы по сравнению с временами решения систем методом Ланцоша. Вместе с тем, для указанных модулей логарифмирования за счет сжатия матриц с последующей перестановкой ненулевых элементов удалось более чем на порядок сократить времена решения линейных систем.

Автор благодарит Н. Л. Замарашкина, М. М. Глухова и А. М. Зубкова за обсуждение результатов работы и ценные замечания.

Таблица 15. Времена решения систем для задачи DL100 на кластере Pentium / Myrinet

P	$S_{(55\,000)}^{(100)}$	$S_{(65\,000)}^{(100)}$	$S_{(75\,000)}^{(100)}$	$S^{(100)}$
2	717.001	280.303	217.083	1650.990
4	371.372	147.315	117.156	949.232
8	179.322	86.477	66.239	619.208
16	96.089	48.632	43.614	409.780
20	71.775	40.290	37.791	374.177
24	63.998	34.450	33.718	348.348
32	48.422	28.004	27.552	310.476
40	40.127	24.022	24.197	286.916
48	37.484	23.454	24.015	313.967
56	32.732	21.657	22.781	310.127
64	29.654	20.095	21.395	304.892
72	27.041	18.777	20.385	297.911
80	24.551	17.920	19.302	286.043
88	23.222	16.800	18.458	284.647
90	22.797	16.575	18.291	283.705

Таблица 16. Времена решения систем для задачи DL100 на кластере Opteron / InfiniBand

P	$S_{(55\,000)}^{(100)}$	$S_{(65\,000)}^{(100)}$	$S_{(75\,000)}^{(100)}$	$S^{(100)}$
2	172.875	73.881	62.588	584.243
4	97.721	42.571	37.337	354.814
8	56.659	27.466	24.047	237.389
16	30.088	15.941	15.814	158.221
20	29.842	15.164	15.356	160.909
24	25.543	14.528	15.244	155.154
32	21.032	11.527	12.753	128.980
40	15.442	10.950	13.042	124.333
48	16.591	10.083	14.727	141.826
56	16.401	9.410	13.230	142.896
64	15.911	9.635	13.501	154.179
72	14.285	10.010	15.239	145.545
80	14.370	11.027	13.819	144.806
88	12.771	11.451	12.712	134.041
90	11.899	10.286	10.364	133.245

Таблица 17. Времена решения систем для задачи DL135 на кластере Pentium / Myrinet

P	$S_{(500\,000)}^{(135)}$	$S_{(564\,000)}^{(135)}$	$S_{(628\,000)}^{(135)}$	$S_{(692\,000)}^{(135)}$	$S^{(135)}$
8	17706.250	9229.128	8094.105	7342.376	—
16	8560.694	5692.117	4380.765	4479.290	—
20	7540.138	4130.046	4078.278	3959.008	28835.077
24	6390.555	3476.015	3557.038	3396.951	27249.755
32	4806.666	3190.882	2971.951	3062.484	24453.207
40	3950.555	2764.644	2568.171	2501.323	22697.071
48	3686.666	2696.546	2592.825	2714.562	25212.694
56	3411.250	2539.044	2479.553	2607.942	25207.047
64	3013.611	2348.537	2333.485	2506.962	24098.874
72	2771.250	2161.060	2215.793	2406.750	23795.362
80	2544.861	2076.042	2113.452	2320.634	22856.591
88	2369.027	1986.846	2029.021	2240.157	22740.833
90	2335.138	1974.626	2020.299	2204.404	22399.205

Таблица 18. Времена решения систем для задачи DL135 на кластере Opteron / InfiniBand

P	$S_{(500\,000)}^{(135)}$	$S_{(564\,000)}^{(135)}$	$S_{(628\,000)}^{(135)}$	$S^{(135)}$
4	11030.228	6047.812	4993.901	—
8	5997.731	3609.905	2854.421	18935.576
16	2799.032	1909.069	1677.021	12604.821
20	2346.157	1543.673	1411.289	11772.120
24	1938.673	1381.343	1253.580	11354.710
32	1485.232	1115.026	1058.837	10019.291
40	1281.431	997.849	997.102	9096.558
48	1121.114	894.404	915.280	9431.957
56	1070.212	836.118	909.966	8949.339
64	980.959	792.155	875.916	9006.267
72	892.904	752.872	846.570	8641.488
80	847.694	736.812	847.430	8365.086
88	832.054	693.795	801.985	8959.942
90	830.911	748.903	805.994	8288.098

Таблица 19. Времена решения линейных систем для задач DL100 и DL135

	Pentium / Myrinet			
	DL100		DL135	
P	$T(P)$	$\frac{E(P)}{E(8)}$	$T(P)$	$\frac{E(P)}{E(8)}$
8	66.239	1.000	7342.376	1.000
16	43.614	0.759	4380.765	0.838
20	37.791	0.701	3959.008	0.741
24	33.718	0.654	3396.951	0.720
32	27.552	0.601	2971.951	0.617
40	24.022	0.551	2501.323	0.587
48	23.454	0.470	2592.825	0.471
56	21.657	0.436	2479.553	0.423
64	20.095	0.412	2333.485	0.393
72	18.777	0.391	2161.060	0.377
80	17.920	0.369	2076.042	0.353
88	16.800	0.358	1986.846	0.335
90	16.575	0.355	1974.626	0.330

	Opteron / InfiniBand			
	$p^{(100)}$		$p^{(135)}$	
P	$T(P)$	$\frac{E(P)}{E(4)}$	$T(P)$	$\frac{E(P)}{E(4)}$
4	37.337	1.000	4993.901	1.000
8	24.047	0.776	2854.421	0.874
16	15.814	0.590	1677.021	0.744
20	15.164	0.492	1411.289	0.707
24	14.528	0.428	1253.580	0.663
32	11.527	0.404	1058.837	0.589
40	10.950	0.340	997.102	0.500
48	10.083	0.308	894.404	0.465
56	9.410	0.283	836.118	0.426
64	9.635	0.242	792.155	0.394
72	10.010	0.207	752.872	0.368
80	11.027	0.169	736.812	0.338
88	11.451	0.148	693.795	0.327
90	10.286	0.161	748.903	0.296

Список литературы

1. *Diffie W., Hellman M.E.* New directions in cryptography // IEEE Trans. Inform. Theory. — 1976. — V. 22. № 6. — P. 644–654.
2. *Odlyzko A.M.* Discrete logarithms: the past and future // Designs, Codes and Cryptography. — 2000. — V. 19. № 2. — P. 129–145.
3. *LaMacchia B.A., Odlyzko A.M.* Computation of discrete logarithms in prime fields // Designs, Codes and Cryptography. — 1991. — V. 1. № 1. — P. 47–62.
4. *Weber D.* On the computation of discrete logarithms in finite prime fields, Ph. D. Thesis // Univ. des Saarlandes. — 1997. — 111 p.
5. *Denny T.F.* Lösen großer dünnbesetzter Gleichungssysteme über endlichen Primkörpern, Ph. D. Thesis // Univ. des Saarlandes. — 1997. — 137 p.
6. *Joux A., Lercier R.* Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the Gaussian integer method // Mathematics of Computation. — 2003. — V. 72. № 242. — P. 953–967.
7. *Joux A., Lercier R.* Discrete logarithms in $GF(p)$. — <http://listserv.nodak.edu/archives/nmbrthry.html>, 19.01.2001.
8. *Joux A., Lercier R.* Discrete logarithms in $GF(p)$. — <http://listserv.nodak.edu/archives/nmbrthry.html>, 17.04.2001.
9. *Joux A., Lercier R.* Discrete logarithms in $GF(p)$. — 130 digits. — <http://listserv.nodak.edu/archives/nmbrthry.html>, 18.06.2005.
10. *Dorofeev A. Ya., Dygin D.M., Matyukhin D.V.* Discrete logarithms in $GF(p)$ — 135 digits // International Scientific Conference «Diophante and analytic problems in number theory» dedicated to the 100th birthday of A.O. Gelfond: Abstracts, Moscow, 2007. — P. 14–15. — <http://listserv.nodak.edu/archives/nmbrthry.html>, 22.12.2006.
11. *Kleinjung T.* Discrete logarithms in $GF(p)$. — 160 digits. — <http://listserv.nodak.edu/archives/nmbrthry.html>, 05.02.2007.
12. *Lanczos C.* Solutions of systems of linear equations by minimized iterations // J. Res. Nat. Bureau of Standards. — 1952. — V. 49. — P. 33–53.
13. *Wiedemann D.H.* Solving sparse linear equations over finite fields // IEEE Trans. Inform. Theory. — 1986. — V. 32. № 4. — P. 354–375.
14. *LaMacchia B.A., Odlyzko A.M.* Solving large sparse linear systems over finite fields // Lecture Notes in Computer Science. — 1991. — V. 537. — P. 109–133.

15. *Lambert R.* Computational aspects of discrete logarithms, Ph.D. Thesis // University of Waterloo, Ontario: 1996. — 113 p.
16. *Odlyzko A.M.* Discrete logarithms in finite fields and their cryptographic significance // Lecture Notes in Computer Sciences. — 1984. — V. 209. — P. 224–316.
17. *Coppersmith D., Odlyzko A.M., Schroepel R.* Discrete logarithms in $GF(p)$ // Algorithmica. — 1986. — V. 1, № 1. — P. 1–15.
18. *Матюхин Д. В.* Об асимптотической сложности дискретного логарифмирования в поле $GF(p)$ // Дискретная математика. — 2003. — Т. 15. Вып. 1. — С. 28–49.
19. *Василенко О. Н.* Теоретико-числовые алгоритмы в криптографии. — М.: МЦНМО, 2003. — 325 с.
20. *Pohlig S., Hellman M.* An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance // IEEE Trans. Inform. Theory. — 1978. — V. 24. — P. 106–110.
21. *Gordon D. M.* Discrete logarithms in $GF(p)$ using the number field sieve // SIAM J. Discr. Math. — 1993. — V. 6, № 1. — P. 124–138.
22. *Semaev I.* Special prime numbers and discrete logs in prime finite fields // Mathematics of Computation. — 2002. — V. 71, № 237. — P. 363–377.
23. *Гантмахер Ф. Р.* Теория матриц. — М., 1988. — 491 с.
24. *Дорофеев А. Я.* Вычисление логарифмов в конечном простом поле методом линейного решета // Труды по дискретной математике. — 2002. — Т. 5. — С. 29–50.
25. *Eberly W., Kaltofen E.* On randomized Lanczos algorithms // Proc. Internat. Symp. Symbolic Algebraic Comput., ISSAC '97. — N. Y., 1997. — P. 176–183.
26. *Тьюарсон Р.* Разреженные матрицы. — М.: Мир, 1977. — 189 с.
27. *Кнут Д. Е.* Искусство программирования. Том 2. Получисленные алгоритмы. — М.: Вильямс, 2000. — 828 с.
28. *Barret R. et al.* Templates for the solution of linear systems: building blocks for iterative methods. — Philadelphia: SIAM, 1994. — 107 p.
29. *Duff I. S., van der Vorst H. A.* Developments and trends in the parallel solution of linear systems // Parallel Computing. — 1999. — V. 25. — P. 1931–1970.
30. *Markowitz H. M.* The elimination form of the inverse and its application to linear programming // Management Sci. — 1957. — № 3. — P. 255–269.
31. *Кнут Д. Е.* Искусство программирования. Том 3. Сортировка и поиск. — М.: Вильямс, 2000. — 822 с.
32. *Dehn T., Eiermann M., Giebertmann K., Sperling V.* Structured sparse matrix-vector multiplication on massively parallel SIMD architectures // Parallel Computing. — 1995. — V. 21. — P. 1867–1894.

33. *McLay R. T., Swift S., Carey G. F.* Maximizing sparse matrix-vector performance on RISC based MIMD computers // *Journal of Parallel and Distributed Computing.* — 1996. — V. 37. — P. 146–158.
34. *Ujaldon M., Zapata E. L., Sharma S. D., Saltz, J.* Parallelization techniques for sparse matrix applications // *Journal of Parallel and Distributed Computing.* — 1996. — V. 38. — P. 256–266.
35. *Colombet L., Michallon P., Trystram D.* Parallel matrix-vector product on rings with a minimum of communications // *Parallel Computing.* — 1996. — V. 22. — P. 289–310.
36. *Basermann A.* Conjugate gradient and Lanczos methods for sparse matrices on distributed memory multiprocessor // *Journal of Parallel and Distributed Computing.* — 1997. — V. 45. — P. 46–52.
37. *Nastea A. G., Frieder O., El-Ghazawi T.* Load-balanced sparse matrix-vector multiplication on parallel computers // *Journal of Parallel and Distributed Computing.* — 1997. — V. 46. — P. 180–193.
38. *Basermann A., Reichel B., Schelthoff C.* Preconditioned CG methods for sparse matrices on massively parallel machines // *Parallel Computing.* — 1997. — V. 23. — P. 381–398.
39. *Ghose D., Kim H. J.* Load partitioning and trade-off study for large matrix-vector computations in multicast bus networks with communication delays // *Journal of Parallel and Distributed Computing.* — 1998. — V. 55. — P. 32–59.
40. *Geus R., Rollin S.* Towards a fast parallel sparse symmetric matrix-vector multiplication // *Parallel Computing.* — 2001. — V. 27. — P. 883–896.
41. *Gropp W., Lusk E., Skjellum A.* Using MPI. Portable Parallel Programming with the Message-Passing Interface. — MIT Press, 1999. — 371 p.
42. *Корнеев В. Д.* Параллельное программирование в MPI. — М., 2003. — 303 с.
43. *Воеводин В. В., Воеводин В. В.* Параллельные вычисления. — СПб.: 2004. — 608 с.
44. *Забродин А. В., Левин В. К.* Опыт разработки параллельных вычислительных технологий. Создание и развитие семейства МВС // Труды Всероссийской научной конференции «Высокопроизводительные системы и их приложения». — М., 2000. — С. 3–8.
45. *Дорофеев А. Я.* Решение разреженных систем линейных уравнений при вычислении логарифмов в конечном простом поле // Материалы шестой конференции «Математика и безопасность информационных технологий» (МаБИТ–2007). — М., 2008. — С. 121–128.