

МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

**ЧИСЛЕННОЕ РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ  
УРАВНЕНИЙ МЕТОДОМ ИСКЛЮЧЕНИЯ ГАУССА И  
ИТЕРАЦИОННЫМИ МЕТОДАМИ**  
ОТЧЕТ О ПРАКТИКЕ

Студента 3 курса 311 группы  
направления 02.03.02 — Фундаментальная информатика и информационные  
технологии  
факультета КНиИТ  
Аношкина Андрея Алексеевича

Проверил  
Старший преподаватель

\_\_\_\_\_

М. С. Портенко

## СОДЕРЖАНИЕ

1	Work 06.....	3
---	--------------	---

## 1 Work 06

### Задание

Выполните разработку параллельного варианта для одного из итерационных методов — верхней релаксации.

Для тестовой матрицы из нулей и единиц проведите вычислительные эксперименты, результаты занесите в таблицу 1.

Какой из алгоритмов Гаусса или итерационный обладает лучшими показателями ускорения? Заполните таблицу 2.

### Определение задачи решения системы линейных уравнений

Множество  $n$  линейных уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

называется системой линейных уравнений или линейной системой.

В более кратком (матричном) виде система может быть представлена как  $Ax = b$ , где  $A = (a_{ij})$  есть вещественная матрица размера  $n \times n$ , а вектора  $b$  и  $x$  состоят из элементов.

Под задачей решения системы линейных уравнений для заданных матрицы  $A$  и вектора  $b$  обычно понимается нахождение значения вектора неизвестных  $x$ , при котором выполняются все уравнения системы.

### Итерационные методы

Рассмотрим подход к решению систем линейных уравнений  $Ax = b$  с невырожденной квадратной матрицей, при котором используя заданное начальное приближение  $x^0$  строится последовательность приближенных решений  $x^0, x^1, \dots, x^k, \dots$  до тех пор пока приближенное решение не будет найдено с требуемой точностью.

Итерации заканчиваются, когда:

- норма невязки  $\|b - Ax^k\| = \max_{1 \leq i \leq n} |b_i - \sum_{j=1}^n a_{ij}x_j^k| < \varepsilon$  не станет малой;
- погрешность определения компонент решения  $\|x^{k+1} - x^k\| < \varepsilon$ , где через  $\|\cdot\|$  обозначена любая векторная норма, не станет малой;

- достигнуто максимальное число итераций  $N$ , на которое готов пойти исследователь;
- перечисленные критерии могут совмещаться

## Метод верхней релаксации

Модификация метода Зейделя:

$$x_i^{k+1} = (1-\omega)x_i^k + \omega \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^n a_{ij}x_j^k}{a_{ii}}, i = 1, 2, \dots, n; k = 0, 1, \dots$$

## Последовательная реализация

Фрагмент кода решения приведен ниже:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <conio.h>
4  #include <time.h>
5  #include <math.h>
6  #include <algorithm>
7
8  // Function for simple initialization of the matrix
9  // and the vector elements
10 void DummyDataInitialization(double* pMatrix, double* pVector, int Size) {
11     for (int i = 0; i < Size; ++i) {
12         pVector[i] = i + 1.0;
13         for (int j = 0; j < Size; ++j)
14             if (j <= i)
15                 pMatrix[i * Size + j] = 1;
16             else
17                 pMatrix[i * Size + j] = 0;
18     }
19 }
20
21 // Function for random initialization of the matrix
22 // and the vector elements
23 void RandomDataInitialization(double* pMatrix, double* pVector, int Size) {
24     srand(unsigned(clock()));
25     for (int i = 0; i < Size; ++i) {
26         pVector[i] = rand() / double(1000);
27         for (int j = 0; j < Size; ++j)
28             if (j <= i)
29                 pMatrix[i * Size + j] = rand() / double(1000);
30             else
31                 pMatrix[i * Size + j] = 0;
32     }
33 }
```

```

34
35 // Function for memory allocation and definition of the objectselements
36 void ProcessInitialization(double*& pMatrix, double*& pVector, double*& pResult, int& Size) {
37     // Setting the size of the matrix and the vector
38     do {
39         printf("\nEnter size of the matrix and the vector: ");
40         scanf_s("%d", &Size);
41         printf("\nChosen size = %d \n", Size);
42         if (Size <= 0)
43             printf("\nSize of objects must be greater than 0!\n");
44     } while (Size <= 0);
45
46
47     // Memory allocation
48     pMatrix = new double[Size * Size];
49     pVector = new double[Size];
50     pResult = new double[Size];
51
52     // Initialization of the matrix and the vector elements
53     DummyDataInitialization(pMatrix, pVector, Size);
54     //RandomDataInitialization(pMatrix, pVector, Size);
55 }
56
57 // Function for formatted matrix output
58 void PrintMatrix(double* pMatrix, int RowCount, int ColCount) {
59     for (int i = 0; i < RowCount; ++i) {
60         for (int j = 0; j < ColCount; ++j)
61             printf("%7.4f ", pMatrix[i * RowCount + j]);
62         printf("\n");
63     }
64 }
65
66 // Function for formatted vector output
67 void PrintVector(double* pVector, int Size) {
68     for (int i = 0; i < Size; ++i)
69         printf("%7.4f ", pVector[i]);
70 }
71
72 // Function for the execution of Gauss algorithm
73 void ResultCalculation(double* pMatrix, double* pVector, double* pResult, int Size) {
74
75     for (int i = 0; i < Size; pResult[i] = pVector[i] / pMatrix[i * Size + i], ++i);
76     double maxDif = 1e+6;
77     double w = 1.5;
78
79     for (; maxDif > 1e-9;) {
80         maxDif = 0;
81         double* pNew = new double[Size];
82         for (int i = 0; i < Size; ++i) {
83             double sum = 0;

```

```

84         for (int j = 0; j < Size; ++j) {
85             if (j < i)
86                 sum += pMatrix[i * Size + j] * pNew[j];
87             else
88                 if (j > i)
89                     sum += pMatrix[i * Size + j] * pResult[j];
90         }
91         pNew[i] = (1 - w) * pResult[i] + w * (pVector[i] - sum) / pMatrix[i * Size + i];
92     }
93
94     for (int i = 0; i < Size; ++i) {
95         maxDif = std::max(maxDif, fabs(pNew[i] - pResult[i]));
96         pResult[i] = pNew[i];
97     }
98
99     delete[] pNew;
100 }
101 }
102
103 // Function for computational process termination
104 void ProcessTermination(double* pMatrix, double* pVector, double* pResult) {
105     delete[] pMatrix;
106     delete[] pVector;
107     delete[] pResult;
108 }
109
110 // Function for testing the result
111 void TestResult(double* pMatrix, double* pVector, double* pResult, int Size) {
112     /* Buffer for storing the vector, that is a result of multiplication
113     of the linear system matrix by the vector of unknowns */
114     double* pRightPartVector;
115
116     // Flag, that shows wheather the right parts vectors are identical or not
117     int equal = 0;
118     double Accuracy = 1e-3; // Comparison accuracy
119     pRightPartVector = new double[Size];
120     for (int i = 0; i < Size; ++i) {
121         pRightPartVector[i] = 0;
122         for (int j = 0; j < Size; ++j)
123             pRightPartVector[i] += pMatrix[i * Size + j] * pResult[j];
124     }
125
126     for (int i = 0; i < Size; i++)
127         if (fabs(pRightPartVector[i] - pVector[i]) > Accuracy)
128             equal = 1;
129
130     if (equal == 1)
131         printf("\nThe result of the parallel Gauss algorithm is NOT correct. Check your code.");
132     else
133         printf("\nThe result of the parallel Gauss algorithm is correct.");

```

```

134
135     delete[] pRightPartVector;
136 }
137 int main() {
138     double* pMatrix; // The matrix of the linear system
139     double* pVector; // The right parts of the linear system
140     double* pResult; // The result vector
141     int Size; // The sizes of the initial matrix and the vector
142     double start, finish, duration;
143     printf("Upper relaxation algorithm for solving linear systems\n");
144
145     // Memory allocation and definition of objects ' elements
146     ProcessInitialization(pMatrix, pVector, pResult, Size);
147
148     // The matrix and the vector output
149     /*printf("Initial Matrix \n");
150     PrintMatrix(pMatrix, Size, Size);
151     printf("Initial Vector \n");
152     PrintVector(pVector, Size);*/
153
154     // Execution of Gauss algorithm
155     start = clock();
156     ResultCalculation(pMatrix, pVector, pResult, Size);
157     finish = clock();
158     duration = (finish - start) / CLOCKS_PER_SEC;
159
160     // Testing the result
161     TestResult(pMatrix, pVector, pResult, Size);
162
163     // Printing the result vector
164     /*printf("\nResult Vector: \n");
165     PrintVector(pResult, Size);*/
166
167     // Printing the execution time of Gauss method
168     printf("\nTime of execution: %f\n", duration);
169
170     // Computational process termination
171     ProcessTermination(pMatrix, pVector, pResult);
172
173     return 0;
174 }

```

## Параллельная реализация

Фрагмент кода решения приведен ниже:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <conio.h>
4  #include <time.h>

```

```

5  #include <math.h>
6  #include <omp.h>
7  #include <algorithm>
8
9  // Function for simple initialization of the matrix
10 // and the vector elements
11 void DummyDataInitialization(double* pMatrix, double* pVector, int Size) {
12     for (int i = 0; i < Size; ++i) {
13         pVector[i] = i + 1.0;
14         for (int j = 0; j < Size; ++j)
15             if (j <= i)
16                 pMatrix[i * Size + j] = 1;
17             else
18                 pMatrix[i * Size + j] = 0;
19     }
20 }
21
22 // Function for random initialization of the matrix
23 // and the vector elements
24 void RandomDataInitialization(double* pMatrix, double* pVector, int Size) {
25     srand(unsigned(clock()));
26     for (int i = 0; i < Size; ++i) {
27         pVector[i] = rand() / double(1000);
28         for (int j = 0; j < Size; ++j)
29             if (j <= i)
30                 pMatrix[i * Size + j] = rand() / double(1000);
31             else
32                 pMatrix[i * Size + j] = 0;
33     }
34 }
35
36 // Function for memory allocation and definition of the objectselements
37 void ProcessInitialization(double*& pMatrix, double*& pVector, double*& pResult, int& Size) {
38     // Setting the size of the matrix and the vector
39     do {
40         printf("\nEnter size of the matrix and the vector: ");
41         scanf_s("%d", &Size);
42         printf("\nChosen size = %d \n", Size);
43         if (Size <= 0)
44             printf("\nSize of objects must be greater than 0!\n");
45     } while (Size <= 0);
46
47
48     // Memory allocation
49     pMatrix = new double[Size * Size];
50     pVector = new double[Size];
51     pResult = new double[Size];
52
53     // Initialization of the matrix and the vector elements
54     DummyDataInitialization(pMatrix, pVector, Size);

```



```

55         //RandomDataInitialization(pMatrix, pVector, Size);
56     }
57
58     // Function for formatted matrix output
59     void PrintMatrix(double* pMatrix, int RowCount, int ColCount) {
60         for (int i = 0; i < RowCount; ++i) {
61             for (int j = 0; j < ColCount; ++j)
62                 printf("%7.4f ", pMatrix[i * RowCount + j]);
63             printf("\n");
64         }
65     }
66
67     // Function for formatted vector output
68     void PrintVector(double* pVector, int Size) {
69         for (int i = 0; i < Size; ++i)
70             printf("%7.4f ", pVector[i]);
71     }
72
73     // Function for the execution of Gauss algorithm
74     void ParallelResultCalculation(double* pMatrix, double* pVector, double* pResult, int Size) {
75
76         for (int i = 0; i < Size; pResult[i] = pVector[i] / pMatrix[i * Size + i], ++i);
77         double maxDif = 1e+6;
78         double w = 1.5;
79
80         for (; maxDif > 1e-9;) {
81             maxDif = 0;
82             double* pNew = new double[Size];
83             #pragma omp parallel for
84             for (int i = 0; i < Size; ++i) {
85                 double sum = 0;
86                 for (int j = i + 1; j < Size; ++j)
87                     sum += pMatrix[i * Size + j] * pResult[j];
88
89                 pNew[i] = (1 - w) * pResult[i] + w * (pVector[i] - sum) / pMatrix[i * Size + i];
90             }
91
92             for (int i = 0; i < Size; ++i) {
93                 double sum = 0;
94                 for (int j = 0; j < i; ++j)
95                     sum += pMatrix[i * Size + j] * pNew[j];
96                 pNew[i] += w * -sum / pMatrix[i * Size + i];
97             }
98
99             for (int i = 0; i < Size; ++i) {
100                 maxDif = std::max(maxDif, fabs(pNew[i] - pResult[i]));
101                 pResult[i] = pNew[i];
102             }
103
104             delete[] pNew;

```

```

105     }
106 }
107
108 // Function for computational process termination
109 void ProcessTermination(double* pMatrix, double* pVector, double* pResult) {
110     delete[] pMatrix;
111     delete[] pVector;
112     delete[] pResult;
113 }
114
115 // Function for testing the result
116 void TestResult(double* pMatrix, double* pVector, double* pResult, int Size) {
117     /* Buffer for storing the vector, that is a result of multiplication
118     of the linear system matrix by the vector of unknowns */
119     double* pRightPartVector;
120
121     // Flag, that shows wheather the right parts vectors are identical or not
122     int equal = 0;
123     double Accuracy = 1e-3; // Comparison accuracy
124     pRightPartVector = new double[Size];
125     for (int i = 0; i < Size; ++i) {
126         pRightPartVector[i] = 0;
127         for (int j = 0; j < Size; ++j)
128             pRightPartVector[i] += pMatrix[i * Size + j] * pResult[j];
129     }
130
131     for (int i = 0; i < Size; i++)
132         if (fabs(pRightPartVector[i] - pVector[i]) > Accuracy)
133             equal = 1;
134
135     if (equal == 1)
136         printf("\nThe result of the parallel Gauss algorithm is NOT correct. Check your code.");
137     else
138         printf("\nThe result of the parallel Gauss algorithm is correct.");
139
140     delete[] pRightPartVector;
141 }
142 int main() {
143     double* pMatrix; // The matrix of the linear system
144     double* pVector; // The right parts of the linear system
145     double* pResult; // The result vector
146     int Size; // The sizes of the initial matrix and the vector
147     double start, finish, duration;
148     printf("Parallel upper relaxation algorithm for solving linear systems\n");
149
150     // Memory allocation and definition of objects' elements
151     ProcessInitialization(pMatrix, pVector, pResult, Size);
152
153     // The matrix and the vector output
154     /*printf("Initial Matrix \n");

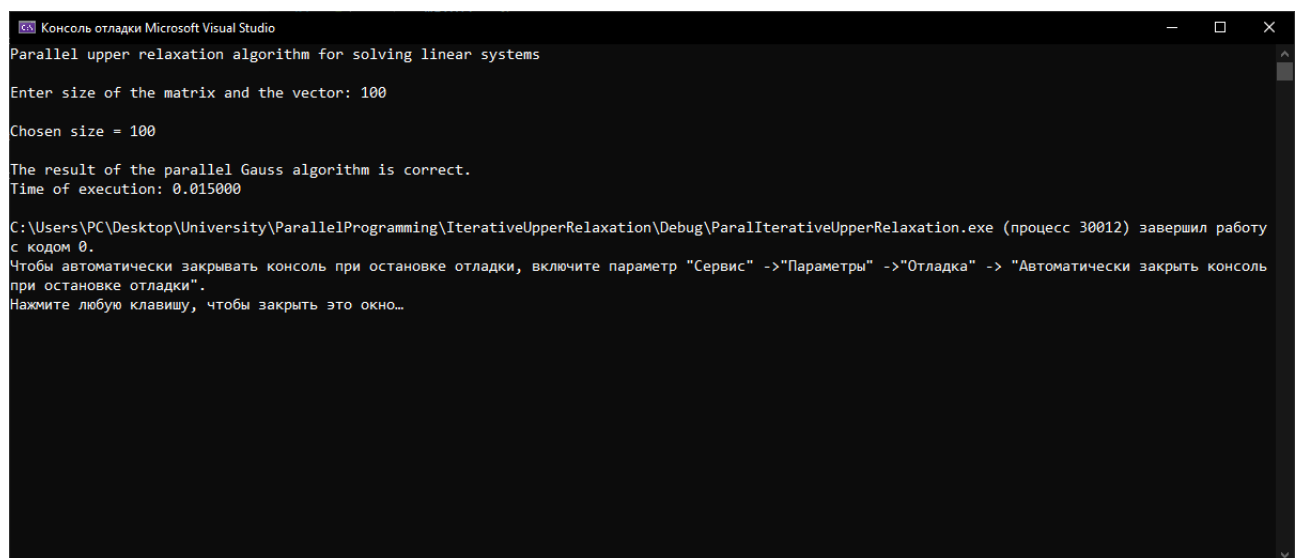
```

```

155     PrintMatrix(pMatrix, Size, Size);
156     printf("Initial Vector \n");
157     PrintVector(pVector, Size);*/
158
159     // Execution of Gauss algorithm
160     start = clock();
161     ParallelResultCalculation(pMatrix, pVector, pResult, Size);
162     finish = clock();
163     duration = (finish - start) / CLOCKS_PER_SEC;
164
165     // Testing the result
166     TestResult(pMatrix, pVector, pResult, Size);
167
168     // Printing the result vector
169     /*printf("\nResult Vector: \n");
170     PrintVector(pResult, Size);*/
171
172     // Printing the execution time of Gauss method
173     printf("\nTime of execution: %f\n", duration);
174
175     // Computational process termination
176     ProcessTermination(pMatrix, pVector, pResult);
177
178     return 0;
179 }

```

## Результат работы



```

Консоль отладки Microsoft Visual Studio
Parallel upper relaxation algorithm for solving linear systems
Enter size of the matrix and the vector: 100
Chosen size = 100
The result of the parallel Gauss algorithm is correct.
Time of execution: 0.015000
C:\Users\PC\Desktop\University\ParallelProgramming\IterativeUpperRelaxation\Debug\ParallIterativeUpperRelaxation.exe (процесс 30012) завершил работу
с кодом 0.
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль
при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно...

```

Рисунок 1 – Work-6

## Таблица сравнения

Номер теста	Порядок системы	Последовательный алгоритм	Параллельный алгоритм	
			Время	Ускорение
1	10	0.000000	0.005000	$\approx \infty$
2	100	0.013000	0.021000	$\approx 0.62$
3	500	1.433000	0.996000	$\approx 1.44$
4	1000	11.130000	7.947000	$\approx 1.40$
5	1500	6.566000	4.659000	$\approx 1.41$
6	2000	11.663000	8.288000	$\approx 1.41$
7	2500	18.223000	13.031000	$\approx 1.40$
8	3000	26.308000	18.698000	$\approx 1.41$

Таблица 1 – Время выполнения последовательного и параллельного итерационного алгоритмов решения систем линейных уравнений и ускорение

Номер теста	Порядок системы	Ускорение алгоритма Гаусса	Ускорение итерационного алгоритма (3)
1	10	$\approx \infty$	$\approx \infty$
2	100	$\approx 0.14$	$\approx 0.62$
3	500	$\approx 1.98$	$\approx 1.44$
4	1000	$\approx 3.59$	$\approx 1.40$
5	1500	$\approx 3.57$	$\approx 1.41$
6	2000	$\approx 4.22$	$\approx 1.41$
7	2500	$\approx 4.36$	$\approx 1.40$
8	3000	$\approx 4.14$	$\approx 1.41$

Таблица 2 – Ускорение параллельных алгоритмов Гаусса и итерационного (3) решения систем линейных уравнений

## Характеристики устройства

Процессор: Intel(R) Core(TM) i5-10400F

Ядер: 6

Оперативная память: 16 Гб