

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI»

студентки 2 курса, группы 21207

Черновской Яны Тихоновны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А.Ю. Власенко

Новосибирск 2023

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ЦЕЛЬ	3
ЗАДАНИЕ	4
ОПИСАНИЕ РАБОТЫ	6
ЗАКЛЮЧЕНИЕ	10
ПРИЛОЖЕНИЕ	11
Приложение 1. Полный листинг последовательной программы на С	11
Приложение 1. Полный листинг параллельной программы на С	15
Приложение 3. Скрипт для запуска последовательной программы	20
Приложение 4. Скрипт для запуска параллельной	21

ЦЕЛЬ

Сравнение времени выполнения решения системы линейных алгебраических уравнений вида $Ax=b$ в соответствии с выбранным вариантом с помощью последовательной и параллельной программы.

ЗАДАНИЕ

1. Написать 2 программы (последовательную и параллельную с использованием MPI) на языке C/C++, которые реализуют итерационный алгоритм решения системы линейных алгебраических уравнений вида $Ax=b$ в соответствии с выбранным вариантом. Здесь A – матрица размером $N \times N$, x и b – векторы длины N . Тип элементов – double.
2. Параллельную программу реализовать с тем условием, что матрица A и вектор b инициализируются на каком-либо одном процессе, а затем матрица A «разрезается» по строкам на близкие по размеру, возможно не одинаковые, части, а вектор b раздается каждому процессу.

Уделить внимание тому, чтобы при запуске программы на различном числе MPI-процессов решалась одна и та же задача (исходные данные заполнялись одинаковым образом).

3. Замерить время работы последовательного варианта программы, а также время работы параллельного при использовании различного числа процессорных ядер. Минимально на 2, 4, 8, 16, 24 (на каждом из наших узлов кластера по 12 ядер), но чем на большем количестве различного числа процессов будут выполнены замеры, тем лучше. Также чем больше замеров будет выполнено на одном и том же числе процессов, тем лучше. В этом случае для построения графиков следует брать минимальное время. Построить графики зависимости времени работы программы, ускорения и эффективности распараллеливания от числа используемых ядер. Исходные данные, параметры N и ϵ подобрать таким образом, чтобы решение задачи на одном ядре занимало не менее 30 секунд.
4. Выполнить профилирование двух вариантов программы с помощью jumpshot или ITAC (Intel Trace Analyzer and Collector) при использовании 16-и или 24-х ядер.
5. На основании полученных результатов сделать вывод.

Вариант задания:

Метод простой итерации

В методе простой итерации преобразование решения на каждом шаге задается формулой:

$$x^{n+1} = x^n - \tau(Ax^n - b).$$

Здесь τ – константа, параметр метода. В зависимости от значения параметра τ последовательность $\{x^n\}$ может сходиться к решению быстрее или медленнее, или вообще расходиться. В качестве подходящего значения τ

¹Общая формула для итерационных методов выглядит следующим образом: $x^{n+1} = f(x^{n+1}, x^n, x^{n-1}, \dots, x^0)$, но для целей лабораторных работ достаточно будет формулы, представленной в тексте.

можно взять 0.01 или -0.01. Знак параметра τ зависит от задачи. Если с некоторым знаком решение начинает расходиться, то следует сменить его на противоположный. Критерий завершения счета:

$$\frac{\|Ax^n - b\|_2}{\|b\|_2} < \varepsilon,$$

где $\|u\|_2 = \sqrt{\sum_{i=0}^{N-1} u_i^2}$. Для тестирования метода значение ε можно взять равным 10^{-5} .

ОПИСАНИЕ РАБОТЫ

1. Была написана последовательная программа, решающая систему линейных алгебраических уравнений

```
hpcuser221@clu:~> qsub default.sh  
5353501.vm-pbs  
hpcuser221@clu:~> qstat 5353501  
Job id          Name        User          Time Use S Queue  
-----          -----        -----          -----  
5353501.vm-pbs  default.sh  hpcuser221  00:00:00 R bl2x220g7_short  
  
hpcuser221@clu:~> cat default.sh.o5353501  
Total time is 35 seconds  
hpcuser221@clu:~> cat default.sh.e5353501  
hpcuser221@clu:~>
```

2. Была написана параллельная реализация решения системы линейных алгебраических уравнений с помощью MPI

- a. На 2 процессах

```
hpcuser221@clu:~> qsub SLAE.sh  
5353502.vm-pbs  
hpcuser221@clu:~> qstat 5353502  
Job id          Name        User          Time Use S Queue  
-----          -----        -----          -----  
5353502.vm-pbs  SLAE.sh   hpcuser221  00:00:00 R bl2x220g7_short  
  
hpcuser221@clu:~> cat SLAE.sh.o5353502  
Number of MPI process: 2  
Total time is 15.943880  
hpcuser221@clu:~> cat SLAE.sh.e5353502  
hpcuser221@clu:~>
```

- b. На 4 процессах

```
hpcuser221@clu:~> qsub SLAE.sh  
5353503.vm-pbs  
  
hpcuser221@clu:~> qstat 5353503  
qstat: Unknown Job Id 5353503.vm-pbs.hpc.nusc.ru  
  
hpcuser221@clu:~> cat SLAE.sh.o5353503  
Number of MPI process: 4  
Total time is 8.195518  
hpcuser221@clu:~> cat SLAE.sh.e5353503  
hpcuser221@clu:~>
```

- c. На 8 процессах

```

hpcuser221@clu:~> qsub SLAE.sh
5353504.vm-pbs
hpcuser221@clu:~> qstat 5353504
Job id          Name           User           Time Use S Queue
-----          -----          -----
5353504.vm-pbs  SLAE.sh       hpcuser221    00:00:00 R bl2x220g7_short

hpcuser221@clu:~> cat SLAE.sh.o5353504
Number of MPI process: 8
Total time is 4.280434
hpcuser221@clu:~> cat SLAE.sh.e5353504
hpcuser221@clu:~>

```

d. Ha 16 pnoqeccax

```

hpcuser221@clu:~> qsub SLAE.sh
5353507.vm-pbs
hpcuser221@clu:~> qstat 5353507
Job id          Name           User           Time Use S Queue
-----          -----          -----
5353507.vm-pbs  SLAE.sh       hpcuser221    00:00:00 R bl2x220g7_short

hpcuser221@clu:~> cat SLAE.sh.o5353507
Number of MPI process: 16
Total time is 4.269075
hpcuser221@clu:~> cat SLAE.sh.e5353507
hpcuser221@clu:~>

```

e. Ha 24 pnoqeccax

```

hpcuser221@clu:~> qsub SLAE.sh
5353509.vm-pbs
hpcuser221@clu:~> qstat 5353509
Job id          Name           User           Time Use S Queue
-----          -----          -----
5353509.vm-pbs  SLAE.sh       hpcuser221    00:00:00 R bl2x220g7_short

hpcuser221@clu:~> cat SLAE.sh.o5353509
Number of MPI process: 24
Total time is 2.966799
hpcuser221@clu:~> cat SLAE.sh.e5353509

```

3. Были построены графики времени, ускорения и эффективности, где

а) Ускорение: $Sp = T_1 / T_p$, где T_1 - время работы последовательной программы, T_p - время работы параллельной программы на p процессах/потоках

б) Эффективность $Ep = Sp / p * 100\%$.

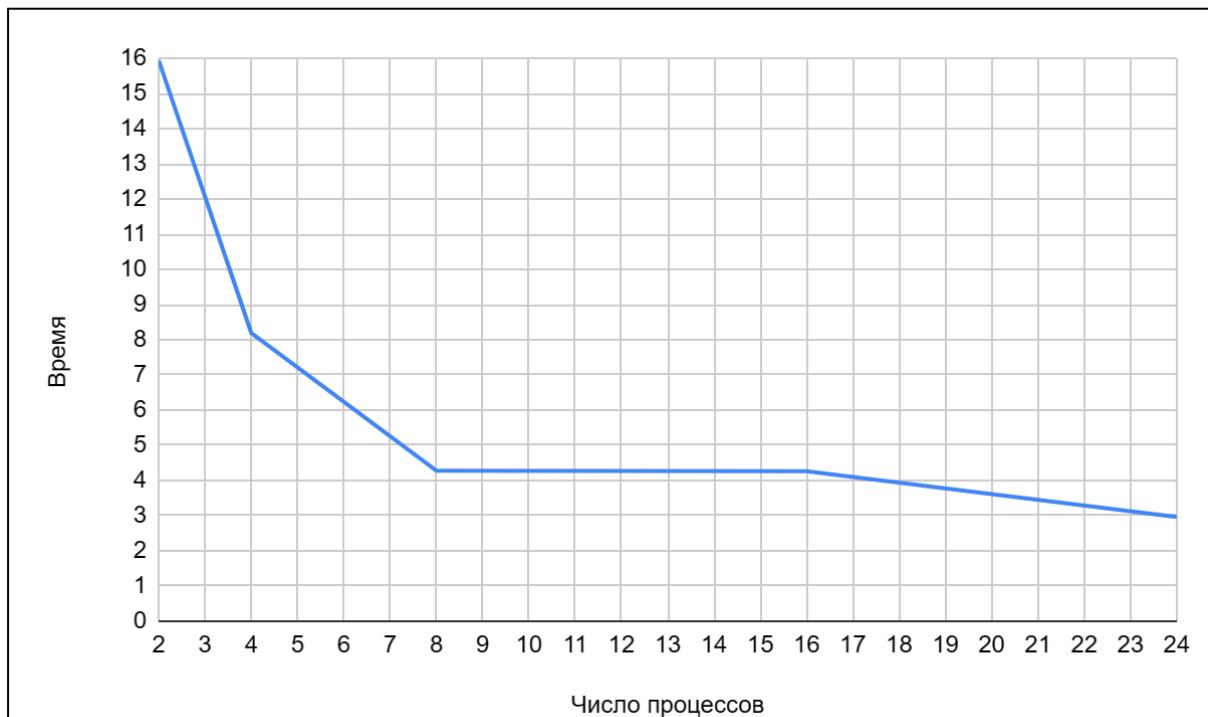


Рис.1 График времени

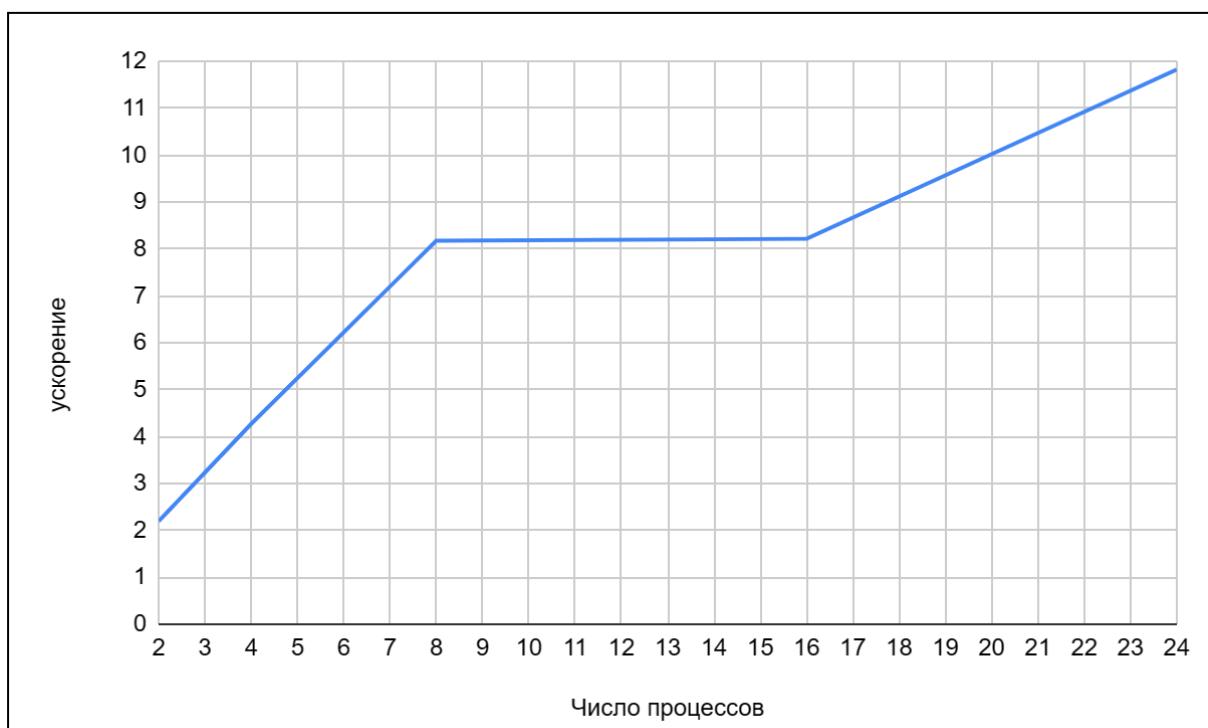


Рис 2. График ускорения

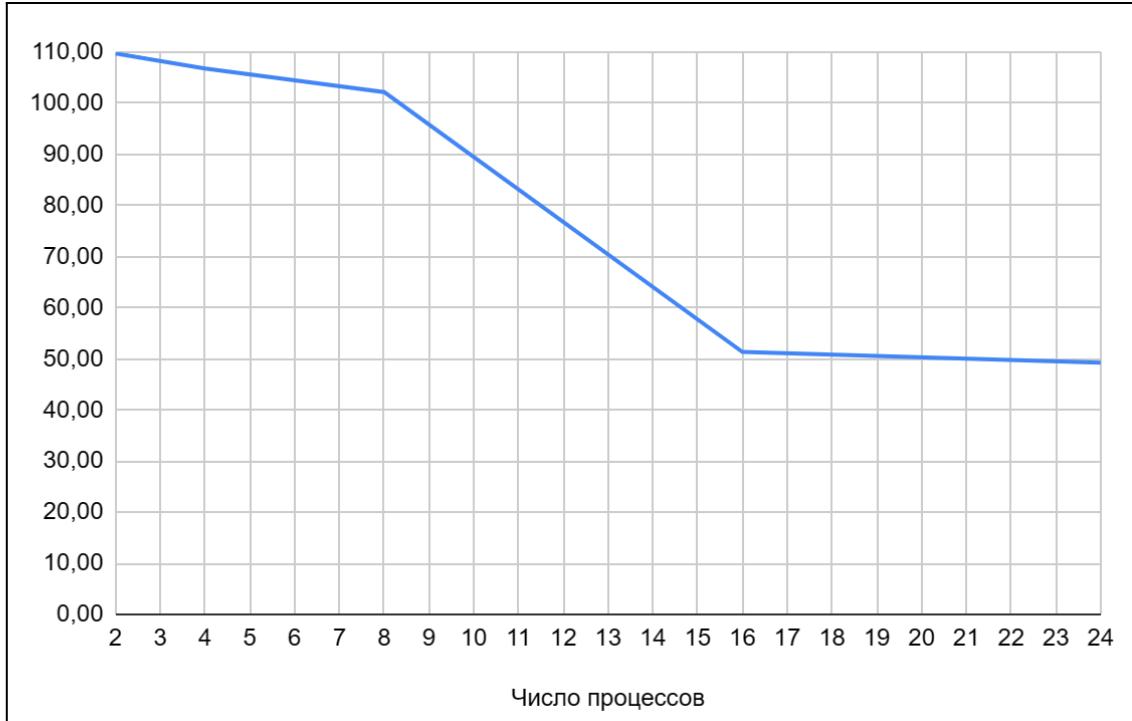
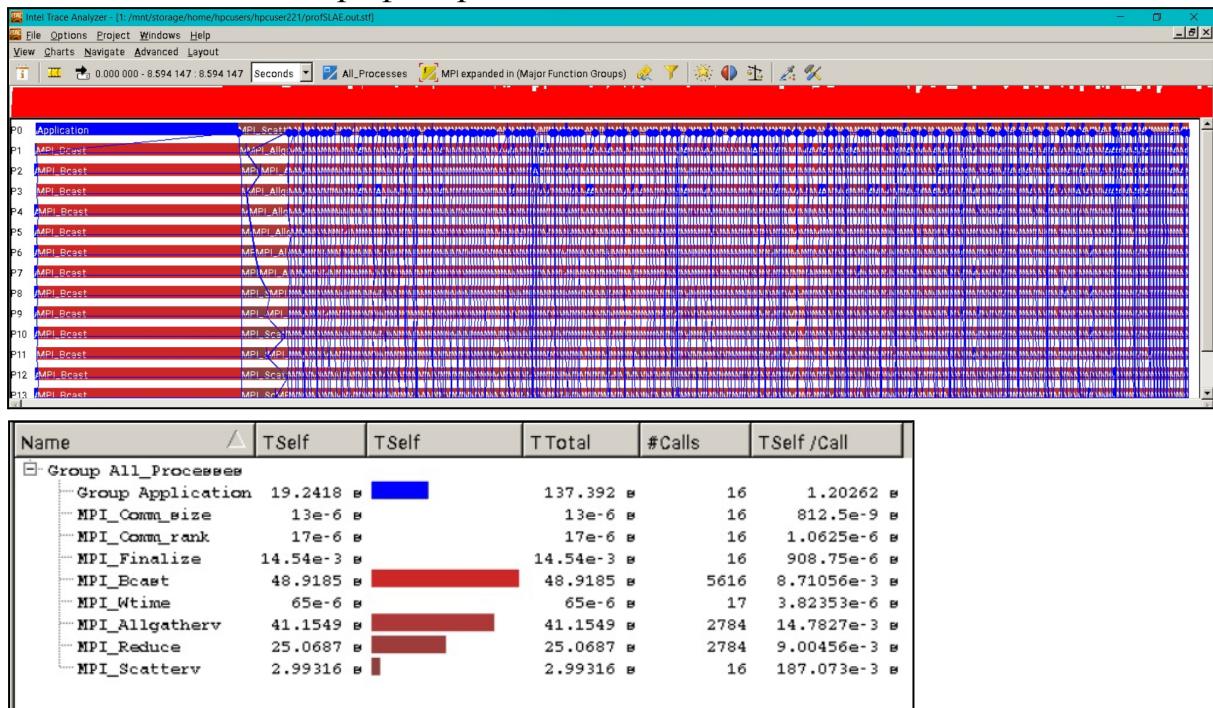
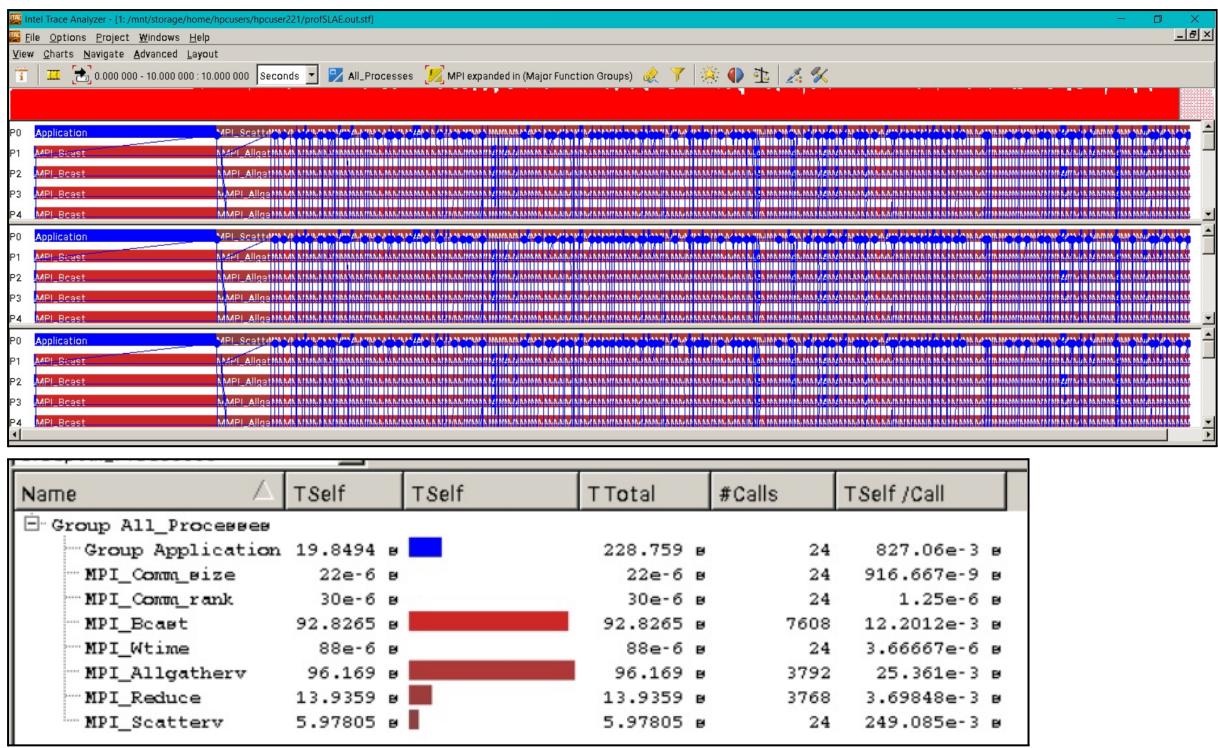


Рис 3. График эффективности

4. Было выполнено профилирование





ЗАКЛЮЧЕНИЕ

Параллельная программа работает в разы быстрее, чем последовательная программа выполнения решения системы линейных алгебраических уравнений вида $Ax=b$ итерационным методом.

ПРИЛОЖЕНИЕ

Приложение 1. Полный листинг последовательной программы на С

```
1. #include <math.h>
2. #include <stdio.h>
3. #include <time.h>
4. #include <stdlib.h>
5.
6. #define N 6000
7. #define EPSILON pow(10, -5)
8. #define MAX_ITERATION_COUNT 50000
9.
10. void generateMatrix(double* matrix);
11. void generateVector(double* vector);
12.
13. void printMatrix(const double* matrix);
14. void printVector(const double* vector);
15.
16. double countNorm(const double* vector);
17. double* subVectors(const double* vector1, const double* vector2, double* result);
18. double* mul(const double* matrix, const double* vector, double* result, int shift);
19. void countNewX(double* x, double* vector);
20.
21. double tau = 0.01;
22. int main(int argc, char **argv)
23. {
24.     srand (time (NULL));
25.
26.     double* A = malloc(sizeof(double) * N * N);
27.     generateMatrix(A);
28.
29.     double* x = malloc(sizeof(double) * N);
30.     generateVector(x);
31.
32.     double* b = malloc(sizeof(double) * N);
33.     generateVector(b);
34.
35.     double* tmpX = malloc(sizeof(double) * N);
36.     int countIterations = 0;
37.     double res = 1;
38.     double prevRes = 0;
39.
40.     time_t begin = time(NULL);
41.     while(res > EPSILON)
42.     {
43.         mul(A, x, tmpX, N);
44.         subVectors(tmpX, b, tmpX); //Ax - b
```

```

45.
46.     res = countNorm(tmpX) / countNorm(b);
47.
48.     countNewX(x, tmpX);
49.     countIterations++;
50.
51.     if (countIterations > MAX_ITERATION_COUNT && prevRes > res)
52.     {
53.         printf("Wrong tau...\n");
54.         free(A);
55.         free(x);
56.         free(b);
57.         return EXIT_SUCCESS;
58.     }
59.     prevRes = res;
60. }
61.
62. time_t end = time(NULL);
63.
64. printf("Total time is %ld seconds\n", (end - begin));
65. free(A);
66. free(x);
67. free(b);
68. free(tmpX);
69.
70. return EXIT_SUCCESS;
71. }
72.
73. void countNewX(double* x, double* vector)
74. {
75.     for (int i = 0; i < N; i++)
76.     {
77.         x[i] = x[i] - tau * vector[i];
78.     }
79. }
80.
81. void generateVector(double* vector)
82. {
83.     for (int i = 0; i < N; i++)
84.     {
85.         vector[i] = (double)rand() / RAND_MAX * 10.0 - 5.0;
86.     }
87. }
88.
89. void generateMatrix(double* matrix)
90. {
91.     for (int i = 0; i < N; i++)
92.     {

```

```

93.     for(int j = 0; j < i; j++)
94.     {
95.         matrix[i*N + j] = matrix[j*N + i];
96.     }
97.
98.     for(int j = i; j < N; j++)
99.     {
100.         matrix[i*N + j] = (double)rand()/RAND_MAX*2.0-1.0; //float in range -1 to
1
101.         if(i == j) matrix[i*N + j] = matrix[i*N + j] + N;
102.
103.     }
104. }
105. }
106.
107. void printMatrix(const double* matrix)
108. {
109.     for (int i = 0; i < N; i++)
110.     {
111.         for (int j = 0; j < N; j++)
112.         {
113.             printf("%f ", matrix[i * N + j]);
114.         }
115.
116.         printf("\n");
117.     }
118.
119.     printf("\n");
120. }
121.
122. void printVector(const double* vector)
123. {
124.     for(int i = 0; i < N; i++)
125.     {
126.         printf("%f ", vector[i]);
127.     }
128.
129.     printf("\n");
130. }
131.
132. double* mul(const double* matrix, const double* vector, double* result, int
shift)
133. {
134.     for (int i = 0; i < shift; i++)
135.     {
136.         result[i] = 0;
137.
138.         for (int j = 0; j < shift; j++)

```

```
139.      {
140.          result[i] += matrix[i * shift + j] * vector[j];
141.      }
142.  }
143.
144. }
145.
146. double* subVectors(const double* vector1, const double* vector2, double*
    result)
147. {
148.     for (int i = 0; i < N; i++)
149.     {
150.         result[i] = vector1[i] - vector2[i];
151.     }
152. }
153.
154. double countNorm(const double* vector)
155. {
156.     double normValue = 0;
157.     for (int i = 0; i < N; i++)
158.     {
159.         normValue += vector[i] * vector[i];
160.     }
161.
162.     return sqrt(normValue);
163. }
164.
```

Приложение 1. Полный листинг параллельной программы на С

```
1. #include <math.h>
2. #include <mpi.h>
3. #include <stdio.h>
4. #include <stdlib.h>
5.
6. #define N 6000
7. #define EPSILON pow(10, -5)
8. #define MAX_ITERATION_COUNT 50000
9. #define TAU 0.01
10.
11. void setMatrixPart(int* lineCounts, int* offsets, int* sendCounts, int* displs, int size, int numProc);
12. double countSquareNorm(const double *vector, int size);
13.
14. void generateVector(double* vector);
15. void generateMatrix(double* matrix);
16. void printMatrix(const double* matrix);
17. void printVector(const double* vector, int size);
18.
19. int main(int argc, char **argv)
20. {
21.     int rank, numProc;
22.     double startTime, endTime;
23.
24.     MPI_Init(&argc, &argv);
25.     startTime = MPI_Wtime();
26.     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
27.     MPI_Comm_size(MPI_COMM_WORLD, &numProc);
28.
29.     int* linesNumber = malloc(sizeof(int) * numProc);
30.     int* offsets = malloc(sizeof(int) * numProc);
31.     int* sendCounts = malloc(sizeof(int) * N);
32.     int* displs = malloc(sizeof(int) * N);
33.     setMatrixPart(linesNumber, offsets, sendCounts, displs, N, numProc);
34.
35.     double* partA = malloc(sizeof(double) * linesNumber[rank] * N);
36.     double* A = malloc(sizeof(double) * N * N);
37.     double* x = malloc(sizeof(double) * N);
38.     double* b = malloc(sizeof(double) * N);
39.
40.     double result;
41.     double normB;
42.     if (rank == 0)
43.     {
44.         generateMatrix(A);
45.         generateVector(x);
```

```

46.     generateVector(b);
47.     result = 1;
48.     normB = sqrt(countSquareNorm(b, N));
49. }
50.
51. MPI_Bcast(b, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
52. MPI_Bcast(x, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
53. MPI_Bcast(&result, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
54. MPI_Scatterv(A, sendCounts, displs, MPI_DOUBLE, partA,
55.                 sendCounts[rank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
56.
57. double* buffer = malloc(sizeof(double) * linesNumber[rank]);
58. double* partX = malloc(sizeof(double) * linesNumber[rank]);
59.
60. double sumNorm = 0;
61. int countIteration = 0;
62.
63. while (result > EPSILON && countIteration < MAX_ITERATION_COUNT)
64. {
65.     for (int i = 0; i < linesNumber[rank]; ++i)
66.     {
67.         for (int j = 0; j < N; ++j)
68.             buffer[i] += partA[i * N + j] * x[j];
69.         buffer[i] = buffer[i] - b[offsets[rank] + i];
70.     }
71.
72.     for (int i = 0; i < linesNumber[rank]; ++i)
73.         partX[i] = x[offsets[rank] + i] - TAU * buffer[i];
74.
75.     MPI_Allgatherv(partX, linesNumber[rank], MPI_DOUBLE,
76.                    x, linesNumber, offsets, MPI_DOUBLE, MPI_COMM_WORLD);
77.
78.     double partNorm = countSquareNorm(buffer, linesNumber[rank]);
79.     MPI_Reduce(&partNorm, &sumNorm, 1,
80.                MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
81.     if (rank == 0)
82.     {
83.         result = sqrt(sumNorm) / normB;
84.         countIteration++;
85.     }
86.     MPI_Bcast(&countIteration, 1, MPI_INT, 0, MPI_COMM_WORLD);
87.     MPI_Bcast(&result, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
88. }
89.
90. if (rank == 0)
91. {
92.     if (countIteration < MAX_ITERATION_COUNT)
93.     {

```

```

94.     //printVector(x, N);
95.     endTime = MPI_Wtime();
96.     printf("Total time is %f\n", endTime - startTime);
97. }
98.
99. else
100. {
101.     printf("WRONG TAU\n");
102. }
103. }
104.
105. free(linesNumber);
106. free(offsets);
107. free(x);
108. free(b);
109. free(A);
110. free(partA);
111. free(buffer);
112. free(partX);
113.
114. MPI_Finalize();
115.
116. return 0;
117. }
118.
119.
120. void setMatrixPart(int* lineCounts, int* offsets, int* sendCounts, int* displs, int size,
121. int numProc)
122. {
123.     int offset = 0;
124.     for (int i = 0; i < numProc; ++i)
125.     {
126.         lineCounts[i] = size / numProc;
127.         if (i < size % numProc)
128.         {
129.             ++lineCounts[i];
130.         }
131.
132.         offsets[i] = offset;
133.         offset += lineCounts[i];
134.         sendCounts[i] = lineCounts[i] * N;
135.         displs[i] = offsets[i] * N;
136.     }
137. }
138.
139. double countSquareNorm(const double *vector, int size)
140. {

```

```

141.     double norm_square = 0;
142.     for (int i = 0; i < size; ++i)
143.         norm_square += vector[i] * vector[i];
144.
145.     return norm_square;
146. }
147.
148.
149. void generateVector(double* vector)
150. {
151.     for (int i = 0; i < N; i++)
152.     {
153.         vector[i] = (double)rand() / RAND_MAX * 10.0 - 5.0;
154.     }
155. }
156.
157. void generateMatrix(double* matrix)
158. {
159.     for(int i = 0; i < N; i++)
160.     {
161.         for(int j = 0; j < i; j++)
162.         {
163.             matrix[i*N + j] = matrix[j*N + i];
164.         }
165.
166.         for(int j = i; j < N; j++)
167.         {
168.             matrix[i * N + j] = (double)rand() / RAND_MAX * 2.0 - 1.0; //float in range -1 to 1
169.             if(i == j) matrix[i*N + j] = matrix[i*N + j] + N;
170.
171.         }
172.     }
173. }
174.
175. void printMatrix(const double* matrix)
176. {
177.     for (int i = 0; i < N; i++)
178.     {
179.         for (int j = 0; j < N; j++)
180.         {
181.             printf("%f ", matrix[i * N + j]);
182.         }
183.
184.         printf("\n");
185.     }
186.
187.     printf("\n");
188. }
```

```
189.  
190. void printVector(const double* vector, int size)  
191. {  
192.     for(int i = 0; i < size; i++)  
193.     {  
194.         printf("%f ", vector[i]);  
195.     }  
196.  
197.     printf("\n");  
198. }  
199.  
200.  
201.
```

Приложение 3. Скрипт для запуска последовательной программы

1. `#!/bin/sh`
2. `gcc default.c -o default.out -lm -std=c99`

Приложение 4. Скрипт для запуска параллельной программы

```
1. #!/bin/bash
2.
3. #PBS -l walltime=00:01:00
4. #PBS -l select=1:ncpus=1:mpiprocs=24:mem=2000m,place=scatter
5. #PBS -m n
6.
7. cd $PBS_O_WORKDIR
8.
9. MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
10. echo "Number of MPI process: $MPI_NP"
11.
12. mpirun -hostfile $PBS_NODEFILE -perhost 1 -np $MPI_NP ./SLAE.out
```

Приложение 4. Скрипт для запуска параллельной программы с профилированием

```
#!/bin/bash

#PBS -l walltime=00:01:00
#PBS -l select=1:ncpus=1:mpiprocs=16:mem=2000m,place=scatter
#PBS -m n

cd $PBS_O_WORKDIR

MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
echo "Number of MPI process: $MPI_NP"

mpirun -trace -machinefile $PBS_NODEFILE -np $MPI_NP -perhost 2 ./profSLAE.out
```