Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени
Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине «Разработка программных систем»

Студент:	Новокшанов Евгений Андреевич
Группа:	РК6-66Б
Тип задания:	лабораторная работа
Тема:	MPI
Вариант:	9

Студент _____ Новокшанов Е.А.
подпись, дата Фамилия, И.О.

Преподаватель _____ Козов А.В.
подпись, дата Фамилия, И.О.

Москва, 2023

Содержание

Задание

Вариант 9.

Разработать средствами MPI параллельную программу моделирования распространения электрических сигналов в двумерной прямоугольной сетке RC-элементов (одномерный аналог которых представлен на рис. выше). Метод формирования математической модели - узловой. Метод численного интегрирования - явный Эйлера. Внешнее воздействие - источники тока и напряжения. Количество (кратное 8) элементов по каждой из осей в сетке, временной интервал моделирования - параметры программы. Программа должна демонстрировать ускорение по сравнению с последовательным вариантом. Предусмотреть визуализацию результатов посредством утилиты gnuplot. При этом утилита gnuplot должна вызываться отдельной командой после окончания расчета.

Описание структуры программы и реализованных способов взаимодействия процессов

Программа производит численное решение системы ДУ, описывающей распространение электрических сигналов путем параллельного применения прямого метода Эйлера. Уравнение для получения нового значения напряжения в определенном узле, в одномерной цепочке **RC** элементов имеет вид:

$$U_i^{n+1} = U_i^n - \frac{h_t}{R \cdot C}(U_{i-1}^n + U_{i+1}^n - 2U_i^n) \tag{1}$$

Для двумерного же случая, данное уравнение преображается в следующее:

$$U_{i,j}^{n+1} = U_{i,j}^n - \frac{h_t}{R \cdot C}(U_{i-1,j}^n + U_{i+1,j}^n + U_{i,j-1}^n + U_{i,j+1}^n - 4U_{i,j}^n) \tag{2}$$

, где верхний индекс – момент интегрирования (итерация метода Эйлера), нижние индексы – определяют ряд и столбец, в котором находится данный узел, R, C – известные константы, - временной шаг интегрирования.

Поскольку реализуется решение данной задачи возможностями **MPI**, ставится задача распределения узлов для вычисления между процессами. Таким образом, каждому процессу выделяется для вычислений **rows_per_process** соседних строк матрицы узлов RC элементов, где **rows_per_process** = **nodes_vertical** / **comSize**. Каждый процесс производит вычисления в выделенных ему узлах. Поскольку уравнение 1 предполагает известность значений соседних узлов для вычисления текущего, каждый процесс должен иметь информацию о **nodes_horizontal** значений узлов до и после тех, которые он вычисляет. Для обеспечения данной потребности соседние процессы (процессы обрабатывающие соседние строки матрицы узлов RC элементов) должны обмениваться информацией о вычисленных ими значениях узлов. Для этого в каждом процессе резервируется память на массив текущих значений узлов и значений узлов, рассчитанных на предыдущем шаге размером **rows_per_process** * **nodes_horizontal** + 2 * **nodes_horizontal**, где первое слагаемое отвечает за узлы, вычисляемые данным процессом, а второе за буферы, в которых будут находится значения, вычисленные соседними процессами, необходимые для вычисления узлов в

данном процессе. Для обмена вычисленными значениями внутри процессов используется функция **MPI_Sendrecv**, которая вызывается парно в двух соседних процессах, и осуществляет обмен информации, также данная функция позволяет избежать ситуации блокировки, которая появляется при одновременном вызове **MPI_Send** или **MPI_Recv** в обменивающихся информацией процессах. Визуализация разбиения узлов по процессам и порядка обмена информацией приведена на рисунке 1.

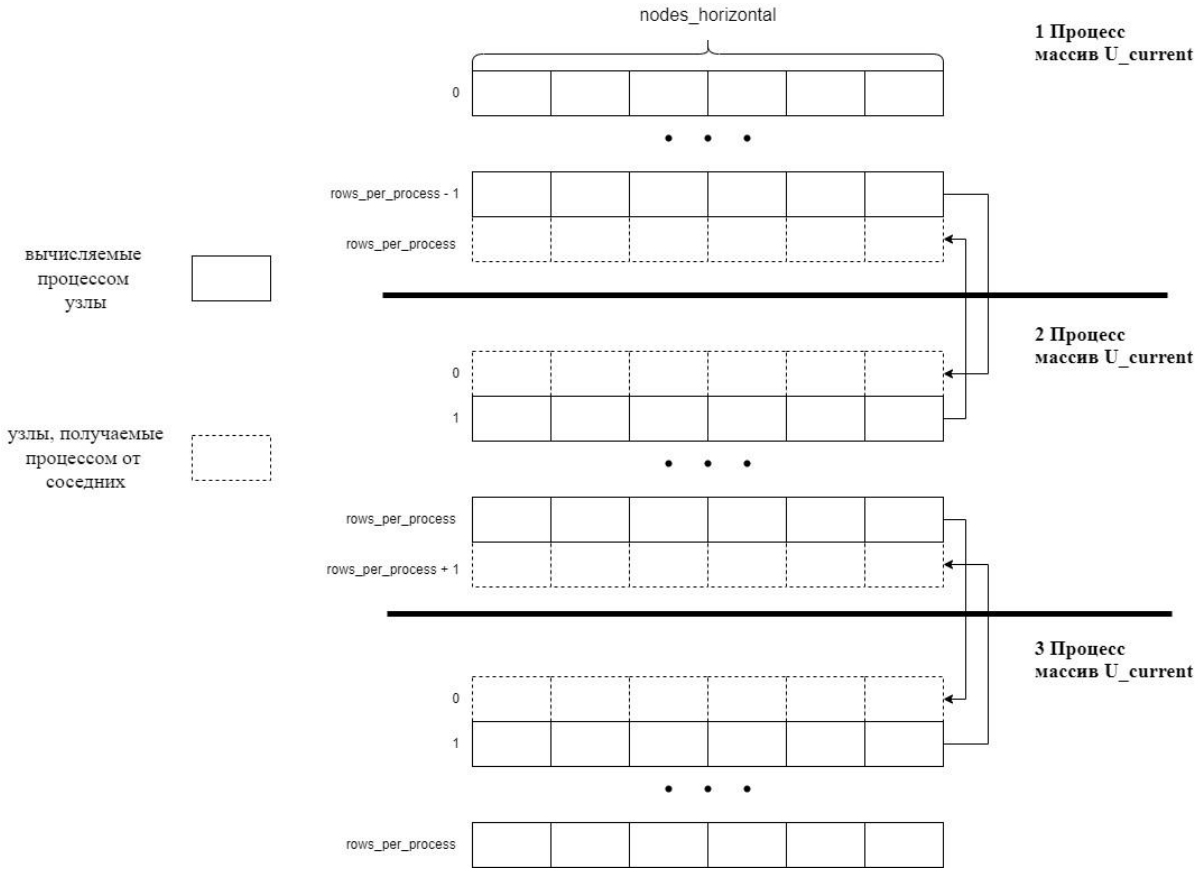

Рис. 1. Граничный обмен между соседними процессами

Для сбора данных и вывода их в файл для последующего отображения на графике используется функция **MPI_Gather**, которая собирает вычисленные значения со всех процессов внутри группы и записывает их в массив процесса **root**. Если же требуется провести проверку времени исполнения программы (без функций вывода информации в файл, сбора информации с процессов..., следует ее компилировать с флагом BENCHMARK=1.

Основные используемые структуры данных

- Макрос BENCHMARK определяет, компилируется ли программа для генерации графиков, или для оценки скорости работы;
- int nodes_vertical – количество RC элементов по вертикали, nodes_horizontal - количество RC элементов по горизонтали, iterations – количество итераций программы;
- double h – временной шаг интегрирования, R – сопротивление резистора, C – емкость конденсатора;
- int time_simulation – время моделирования;
- int boundary_value – значение крайних узлов модели;
- rows_per_process – количество строк матрицы узлов на процесс;
- Переменные comSize и myRank типа int хранят информацию о количестве процессов и идентификаторе текущего процесса;
- double * U_previous, * U_current – массивы, содержащие значения узлов для каждого отдельного процесса (размером rows_per_process * nodes_horizontal + 2 * nodes_horizontal), в предыдущий и текущий момент времени;
- double *U_matrix, FILE *results_file – переменные, необходимые для вывода рассчитанных процессами значений узлов в файл для дальнейшего отображения на графике;
- Структура MPI_Status stat хранит информацию о статусе получения сообщения от соседнего процесса.

Блок-схема

На рисунке 2 представлена блок-схемы программы. Участки кода, которые будут исключены, при компиляции программы с флагом BENCHMARK=1, обозначены на диаграмме прямоугольниками синего цвета с дополнительной идентификацией директивами препроцессора вида #if BENCHMARK == 0, #endif.

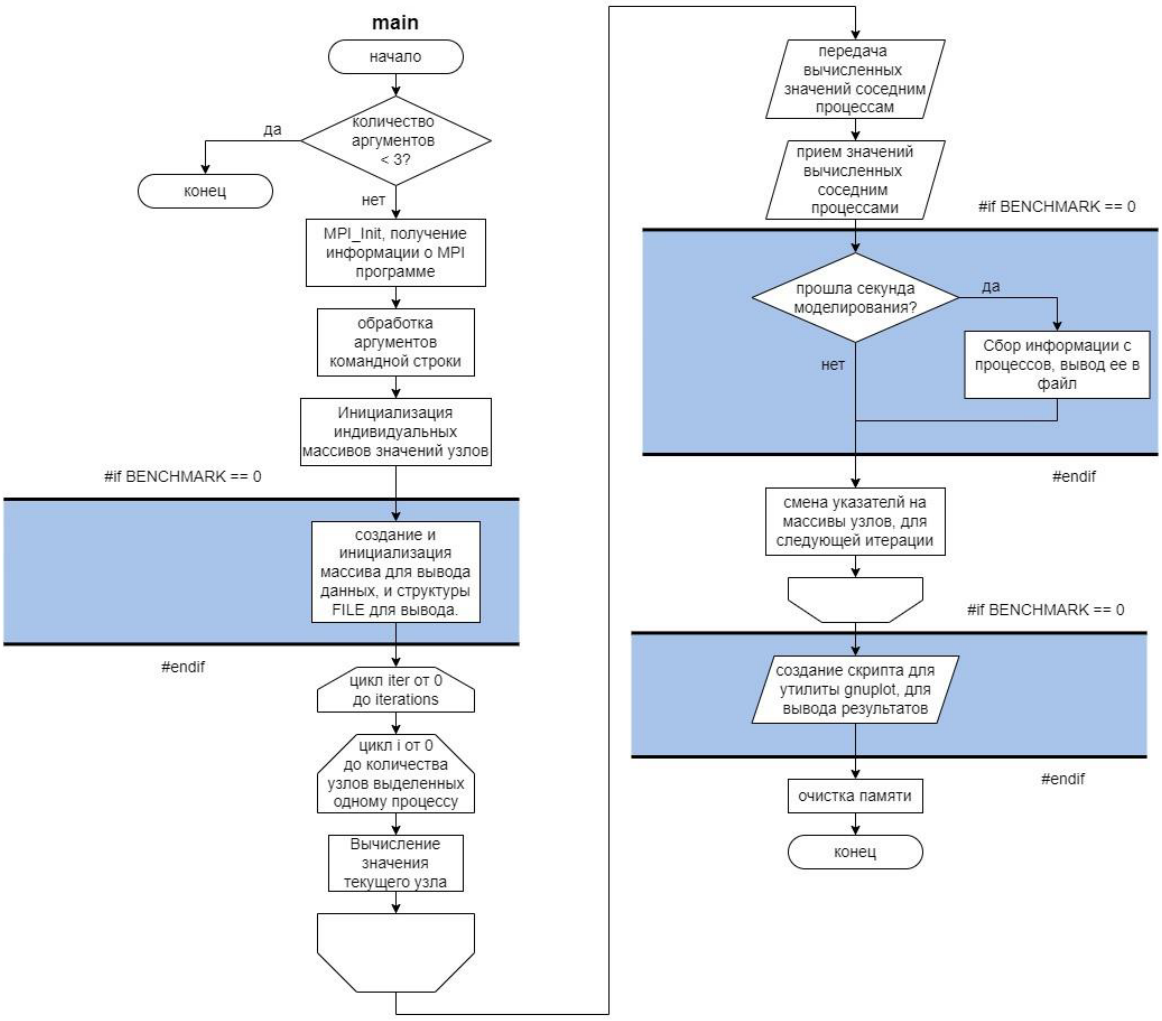

Рис. 2. Блок-схема работы программы 1

Примеры работы программы

На рисунке 3 приведен пример работы программы:вывод 3D графика через gnuplot.

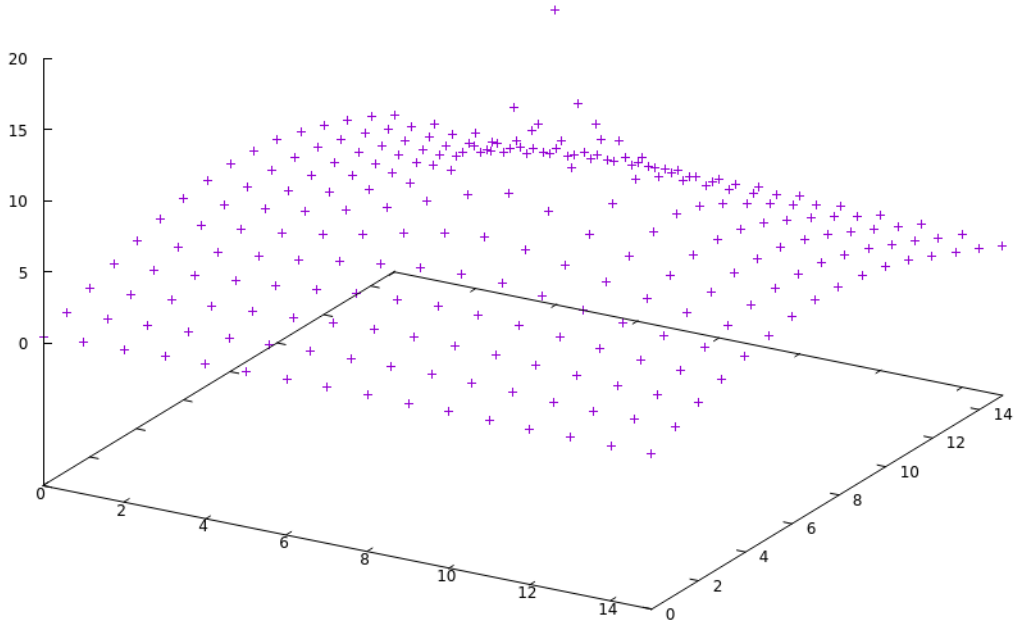

Рис. 3. Пример работы программы, если напряжение подается на центральный элемент

Сравнение результатов работы программы представлено на следующем скриншоте:

```
student11@mgmt-rk6:~$ nano lab4_1.c
student11@mgmt-rk6:~$ mpicc -o lab_zzzz lab4_1.c
student11@mgmt-rk6:~$ mpiexec -f ~/.machinefile -n 1 ./lab_zzzz 1000 1000 1200
Time: 27.693000 seconds
student11@mgmt-rk6:~$ mpiexec -f ~/.machinefile -n 2 ./lab_zzzz 1000 1000 1200
Time: 12.616000 seconds
student11@mgmt-rk6:~$ mpiexec -f ~/.machinefile -n 4 ./lab_zzzz 1000 1000 1200
Time: 7.710000 seconds
student11@mgmt-rk6:~$ mpiexec -f ~/.machinefile -n 8 ./lab_zzzz 1000 1000 1200
Time: 6.121000 seconds
student11@mgmt-rk6:~$ mpiexec -f ~/.machinefile -n 8 ./lab_zzzz 1000 1000 1200
Time: 6.063000 seconds
```

Рис. 4. Пример работы программы, если напряжение подается на центральный элемент

Текст программы

Ниже в листинге 1 представлен текст программы.

Листинг 1. Листинг программы

```
1
2  #include <unistd.h>
3  #include <stdlib.h>
4  #include <stdio.h>
5  #include <string.h>
6  #include <mpi.h>
7  #include <sys/time.h>
8  #ifndef BENCHMARK
9  #define BENCHMARK 0
10 #endif
11
12 struct timeval tv1, tv2, dtv;
13 struct timezone tz;
14
15 void time_start()
16 {
17   gettimeofday(&tv1, &tz);
18 }
19
20
21 int time_stop()
22 {
23   gettimeofday(&tv2, &tz);
24   dtv.tv_sec = tv2.tv_sec - tv1.tv_sec;
25   dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;
26   if (dtv.tv_usec < 0)
27     dtv.tv_sec--; dtv.tv_usec += 1000000;
28   return dtv.tv_sec * 1000 + dtv.tv_usec / 1000;
29 }
30
31
32  int generateGnuplotScript(int nodes_horizontal, int nodes_vertical, int time_simulation);
33
34  int main(int argc, char *argv[]) {
35    //time_t begin = time(NULL);
36    int nodes_vertical = 16, nodes_horizontal = 16, iterations = 0;
37    double h = 0.1, R = 2.5, C = 1;
38    int time_simulation = 1000;
39    int boundary_value = 0;
40
```

```
41    int comSize, myRank;
42    int rows_per_process = 0; // Временной интервал и количество узлов на про-цесс
43
44    // проверка количества аргументов
45    if (argc < 3) {
46      printf("expected to input 3 arguments instead of %d. Aborting.", argc);
47      return 0;
48     }
49
50    MPI_Status stat; // Статус сообщения
51    // Инициализация MPI и получение MPI-переменных
52    MPI_Init(&argc, &argv);
53    MPI_Comm_size(MPI_COMM_WORLD, &comSize);
54    MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
55
56    nodes_vertical = atoi(argv[1]);
57    nodes_horizontal = atoi(argv[2]);
58    iterations = atoi(argv[3]);
59    rows_per_process = nodes_vertical / comSize;
60
61    int print_every = iterations / time_simulation;
62
63    // Выделение массивов для узлов
64    double * U_previous = (double *) malloc((rows_per_process + 2) * nodes_horizontal *
          sizeof(double));
65    memset(U_previous, 0, (rows_per_process + 2) * nodes_horizontal * sizeof(double));
66    double * U_current = (double *) malloc((rows_per_process + 2) * nodes_horizontal *
          sizeof(double));
67    memset(U_current, 0, (rows_per_process + 2) * nodes_horizontal * sizeof(double));
68
69    #if BENCHMARK == 0
70      double *U_matrix;
71      FILE *results_file;
72      if (myRank == 0) {
73      U_matrix = (double *) malloc((nodes_horizontal * nodes_vertical) * sizeof(double));
74      memset(U_matrix, 0, (nodes_horizontal * nodes_vertical) * sizeof(double));
75      results_file = fopen("result.txt", "w");
76      }
77    #endif
78   if (myRank == 0)
79     time_start();
80   for (int iter = 0; iter < iterations; ++iter) {
81     for (int i = 0; i < rows_per_process * nodes_horizontal; ++i) {
82        int cur_node = nodes_horizontal + i;
83        int x = i % nodes_horizontal;
84        int y = myRank * rows_per_process + i / nodes_horizontal;
```

```
85
86        if (y == nodes_horizontal - 1  y == 0  x == 0  x == nodes_vertical - 1) //
              граничный узел
87          U_current[cur_node] = boundary_value;
88
89        else
90
91            U_current[cur_node] = U_previous[cur_node] + h / R / C *
                  (U_previous[cur_node - 1] + U_previous[cur_node + 1] +
                  U_previous[cur_node - nodes_horizontal] + U_previous[cur_node +
                  nodes_horizontal] - 4 * U_previous[cur_node]);
92
93
94  }
95
96   // обмен вычисленной информацией между процессами
97  if (comSize != 1) {
98    if (myRank != 0 && myRank != comSize - 1) {
99      MPI_Sendrecv(U_current + nodes_horizontal, nodes_horizontal, MPI_DOUBLE,
            myRank - 1, 0, U_current, nodes_horizontal, MPI_DOUBLE, myRank - 1, 0,
            MPI_COMM_WORLD, &stat);
100
101     MPI_Sendrecv(U_current + (rows_per_process) * nodes_horizontal,
            nodes_horizontal, MPI_DOUBLE, myRank + 1, 0, U_current +
            (rows_per_process + 1) * nodes_horizontal, nodes_horizontal, MPI_DOUBLE,
            myRank + 1, 0, MPI_COMM_WORLD, &stat);
102   }
103   else if (myRank == 0)
104      MPI_Sendrecv(U_current + (rows_per_process) * nodes_horizontal,
            nodes_horizontal, MPI_DOUBLE, myRank + 1, 0, U_current +
            (rows_per_process + 1) * nodes_horizontal, nodes_horizontal, MPI_DOUBLE,
            myRank + 1, 0, MPI_COMM_WORLD, &stat);
105   else
106      MPI_Sendrecv(U_current + nodes_horizontal, nodes_horizontal, MPI_DOUBLE,
            myRank - 1, 0, U_current, nodes_horizontal, MPI_DOUBLE, myRank - 1, 0,
            MPI_COMM_WORLD, &stat);
107 }
108
109 #if BENCHMARK == 0
110    if (iter % print_every == 0){
111      MPI_Gather(U_current + nodes_horizontal, rows_per_process * nodes_horizontal,
112      MPI_DOUBLE, U_matrix,
113      rows_per_process * nodes_horizontal, MPI_DOUBLE, 0, MPI_COMM_WORLD);
114      if (myRank == 0) { //вывод графика
115       for (int y = 0; y < nodes_vertical; ++y)
116         for (int x = 0; x < nodes_horizontal; ++x)
```

```
117                 fprintf(results_file, "%d %d %f\n", x, y,
118                 U_matrix[y * nodes_horizontal + x]);
119             }
120         }
121
122     #endif
123     double *tmp = U_current;
124     U_current = U_previous;
125     U_previous = tmp;
126     }
127
128      #if BENCHMARK == 0
129         // generate gnuplot script
130         if (myRank == 0) {
131             double ms = time_stop();
132             printf("Time: %f seconds\n", ms/1000);
133             generateGnuplotScript(nodes_horizontal, nodes_vertical, time_simulation);
134             fclose(results_file);
135             free(U_matrix);
136         }
137     #endif
138
139     free(U_current);
140     free(U_previous);
141
142     MPI_Finalize(); // Завершение MPI
143     /*time_t end = time(NULL);
144     printf("time is %ld seconds\n", (end - begin));*/
145
146 }
147
148 int generateGnuplotScript(int nodes_horizontal, int nodes_vertical, int time_simulation){
149     FILE *script_file = fopen("plot_command.gnu", "w+");
150     for (int i = 0; i < time_simulation; i++) {
151         fprintf(script_file, "set samples 35\nset isosamples 35\nset hidden3d offset 3\n");
152         fprintf(script_file, "set xrange[0:%d]\nset yrange[0:%d]\nset zrange[%d:%d]\n",
153       nodes_horizontal - 1, nodes_vertical - 1, 0, 20);
154         fprintf(script_file, "splot %s every ::%d::%d \n", "'result.txt'", i * nodes_horizontal *
155       nodes_vertical,
156         (i + 1) * nodes_horizontal * nodes_vertical);
157         fprintf(script_file, "pause(0.2)\n");
158     }
159     fclose(script_file);
160     return 0;
161 }
```