



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:	Кильдишев Петр Степанович
Группа:	РК6-66Б
Тип задания:	Лабораторная работа №4
Тема:	Программирование средствами МРІ
Вариант:	8

Студент

\_\_\_\_\_  
подпись, дата

Кильдишев П.С.  
Фамилия, И.О.

Преподаватель

\_\_\_\_\_  
подпись, дата

Козов А.В.  
Фамилия, И.О.

Москва, 2023

## Содержание

Задание . . . . .	3
Описание структуры программы и реализованных способов взаимодействия процессов . . . . .	5
Описание основных используемых структур данных . . . . .	7
Блок-схемы . . . . .	8
Примеры работы программы . . . . .	9
Текст программы . . . . .	11

## Задание

Разработать средствами MPI параллельную программу моделирования распространения электрических сигналов в линейной цепочке RC-элементов (см. рис. ниже). Метод формирования математической модели - узловой. Метод численного интегрирования - явный Эйлера. Внешнее воздействие - источники тока и напряжения. Количество (кратное 8) элементов 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.

Узловой метод для формирования математической модели электрической системы использует уравнение второго закона Кирхгофа - сумма токов в узле равна нулю. Это уравнение для любого  $i$ -ого узла в цепочке (кроме крайних) имеет вид  $I_{Rl} - I_{Rr} - I_C = 0$ , где

$I_{Rl} = (U_{i-1} - U_i)/R$  - электрический ток через "левое" сопротивление;

$I_{Rr} = (U_i - U_{i+1})/R$  - электрический ток через "правое" сопротивление;

$I_C = C * dU_i/dt$  - электрический ток через ёмкость;

$U_i$  - электрический потенциал узла с номером  $i$ .

Явный метод Эйлера для численного интегрирования ОДУ подразумевает аппроксимацию производных по времени "разностями вперёд" поэтому дискретизированное уравнение баланса токов в узле принимает следующий вид:

$$(U_{i-1}^n - U_i^n)/R - (U_i^n - U_{i+1}^n)/R - C * (U_i^{n+1} - U_i^n)/ht, \text{ где}$$

$ht$  - величина шага численного интегрирования.

Из него легко выразить единственную неизвестную величину

$$U_i^{n+1} = (U_{i-1}^n - 2 * U_i^n + U_{i+1}^n) * ht / (R * C) - U_i^n$$

Что, в свою очередь, дает возможность просто организовать вычислительный процесс в виде "цикл в цикле" (без деталей, связанных с крайними узлами):

for (n=0; n<N; n++)

for (i=0; i<M; i++)

$$U_i^{n+1} = \dots$$

Напомним, что значения  $U_i^0$  известны из начальных условий. Здесь  $N = T_{end}/ht$ .

## Описание структуры программы и реализованных способов взаимодействия процессов

Для выполнения поставленной задачи реализовано разделение на заданное количество процессов по средствам MPI. Каждому процессу выделяется отрезок узлов, количество элементов в котором равно количеству узлов, деленному на количество процессов. Значения напряжений в этих отрезках хранятся в каждом процессе до конца выполнения программы. Начальные значения рассылаются отцовским (имеющим идентификатор равный 0) процессом в начале выполнения программы после отправки данных о количестве элементов в отрезке и временном интервале. Затем в цикле, считающем до конца временного промежутка, каждому из процессов передаются значения напряжений в узлах, идущих на 1 перед и после назначенного отрезка для вычисления значений внутри отрезка на новом шаге. После вычисления значений на новом шаге, отцовский процесс собирает новые значения на граничных узлах в массивы, чтобы отослать их на следующем шаге. При этом, если требуется визуализация, на каждом шаге происходит вывод в файл значений напряжений в узлах после сбора всех значений в один массив отцовским процессом. Цикл повторяется до тех пор, пока время не достигнет заданного в аргументах запуска, с шагом в 1 секунду. По завершении цикла происходит подсчет времени выполнения программы и вывод его в стандартный поток вывода.

Схема взаимодействия процессов при передаче начальных значений в узлах представлена на рисунке 1.

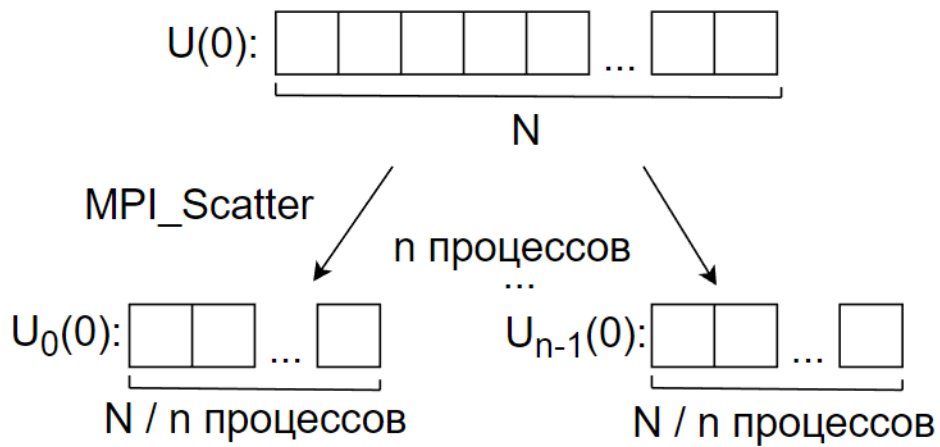


Рис. 1. Схема взаимодействия процессов при передаче начальных значений

Схема взаимодействия процессов в основном цикле до завершения временного интервала представлена на рисунке 2.

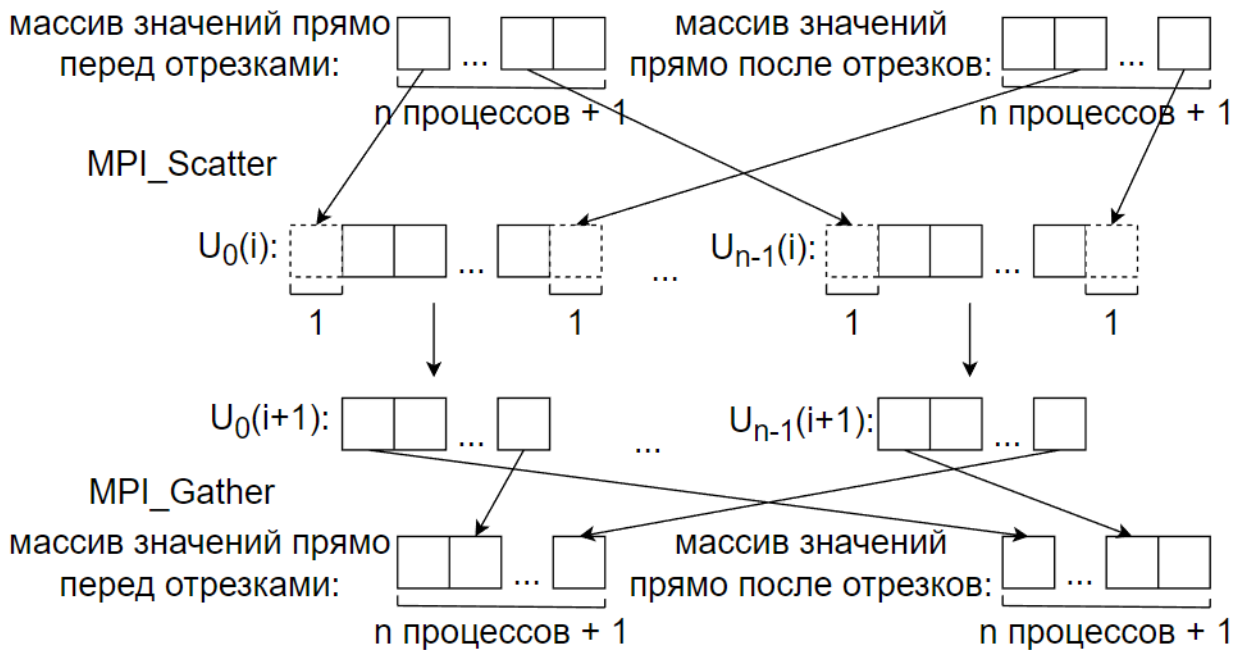


Рис. 2. Схема взаимодействия процессов в основном цикле

## Описание основных используемых структур данных

**long long intBuf[2]** - массив для отправки отцовским процессом дочерним временного интервала и количества узлов в отрезке.

**double\* U\_old** - массив значений напряжений в узлах на одном отрезке на предыдущем шаге.

**double\* U\_new** - массив значений напряжений в узлах на одном отрезке на новом шаге.

**double\* U\_old\_all** - массив значений напряжений во всех узлах.

**double\* U\_first** - массив значений напряжений в узлах прямо перед началами отрезков, при этом запись в него при сборе начинается с 1-го элемента, а 0-м всегда является граничное условие.

**double\* U\_last** - массив значений напряжений в узлах прямо после концов отрезков, при этом считывание из него при рассылке происходит с 1-го элемента, так как в 0-м лежит остаточный мусор, а последним его элементом всегда является граничное условие.

## Блок-схемы

На рисунке 3 представлена блок-схема работы программы.

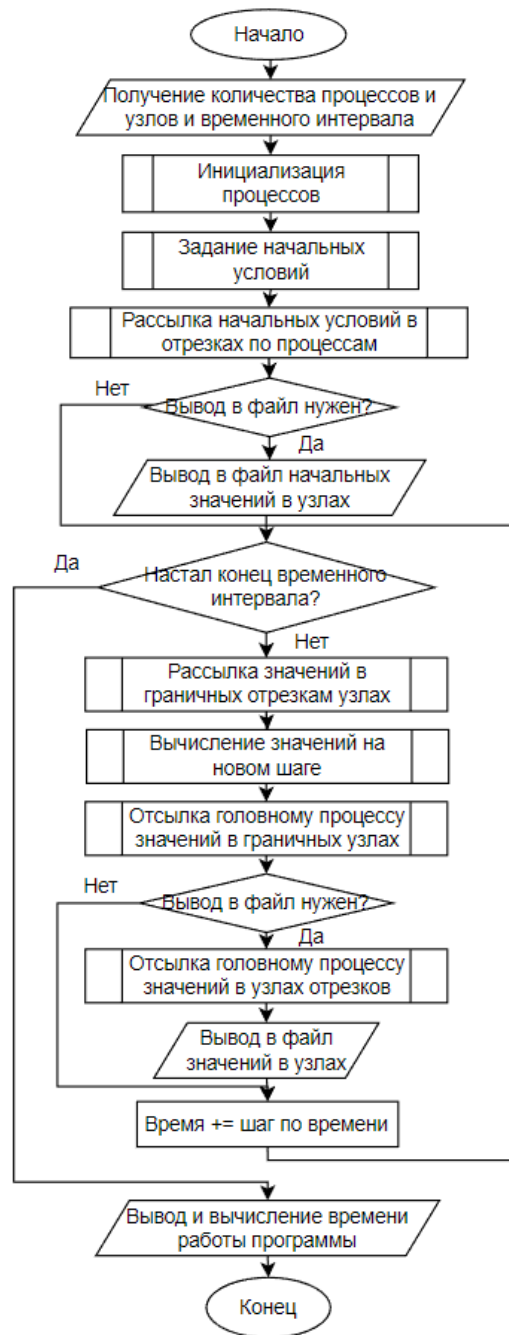
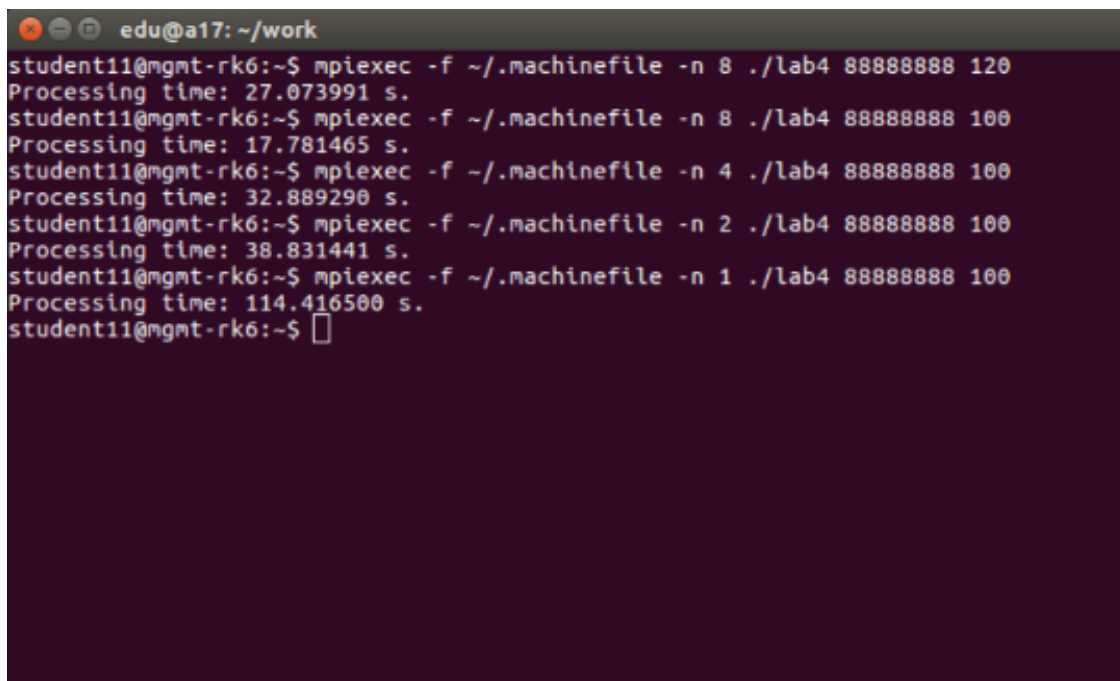


Рис. 3. Блок-схема работы программы



## Примеры работы программы

На рисунке 4 приведен пример работы программы с уменьшением времени выполнения при увеличении количества процессов:

A terminal window with a dark background and light-colored text. The window title is 'edu@a17: ~/work'. The user 'student11@mgmt-rk6' is running a series of MPI commands. Each command is followed by the output 'Processing time: [value] s.'. The commands and their corresponding processing times are: 8 processes (27.073991 s), 8 processes (17.781465 s), 4 processes (32.889290 s), 2 processes (38.831441 s), and 1 process (114.416500 s). The terminal shows that as the number of processes increases, the processing time decreases.

```
edu@a17: ~/work
student11@mgmt-rk6:~$ mpiexec -f ~/.machinofile -n 8 ./lab4 88888888 120
Processing time: 27.073991 s.
student11@mgmt-rk6:~$ mpiexec -f ~/.machinofile -n 8 ./lab4 88888888 100
Processing time: 17.781465 s.
student11@mgmt-rk6:~$ mpiexec -f ~/.machinofile -n 4 ./lab4 88888888 100
Processing time: 32.889290 s.
student11@mgmt-rk6:~$ mpiexec -f ~/.machinofile -n 2 ./lab4 88888888 100
Processing time: 38.831441 s.
student11@mgmt-rk6:~$ mpiexec -f ~/.machinofile -n 1 ./lab4 88888888 100
Processing time: 114.416500 s.
student11@mgmt-rk6:~$
```

Рис. 4. Пример работы программы с уменьшением времени выполнения

На рисунке 5 приведен пример визуализации результата работы программы:

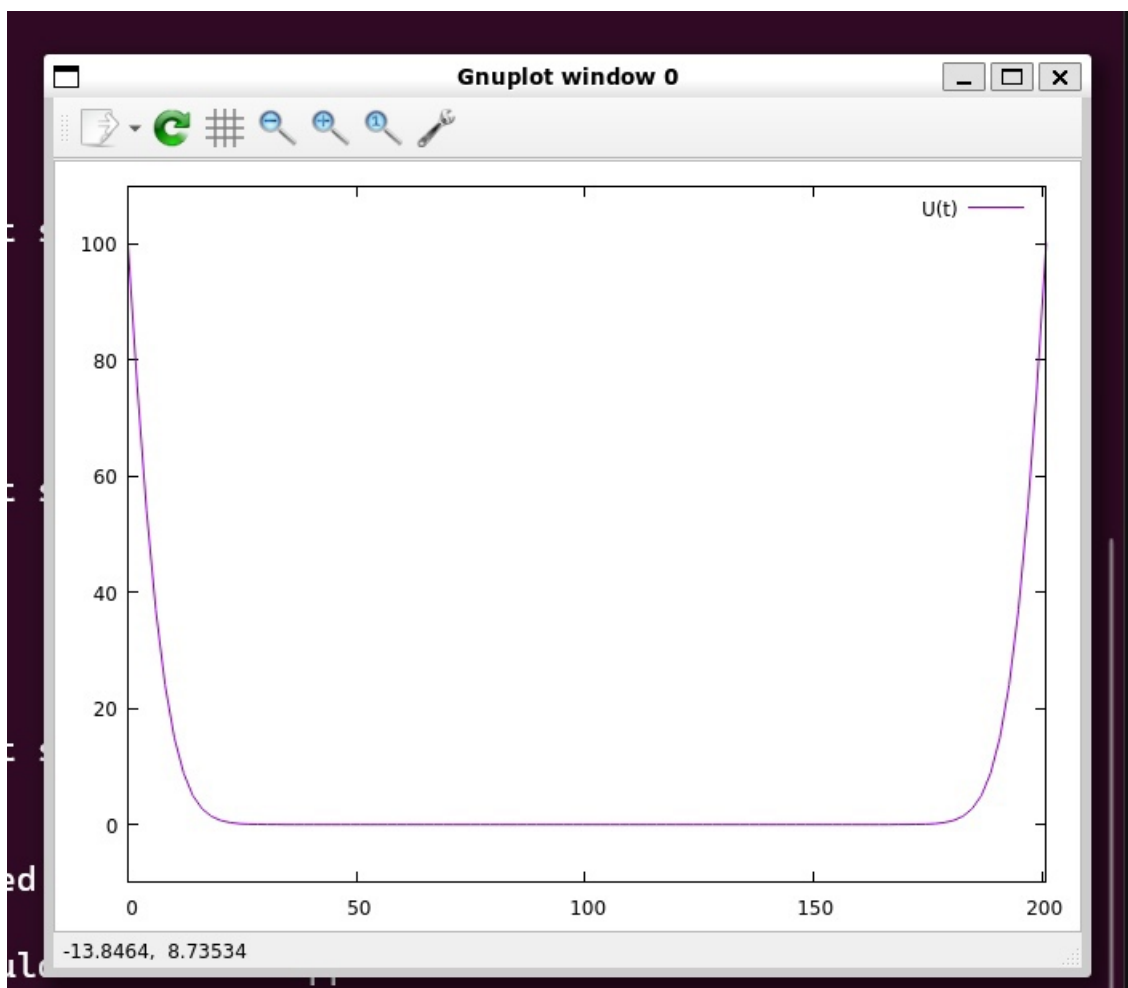


Рис. 5. Пример визуализации результата работы программы

## Текст программы

Ниже в листинге 1 представлен текст программы.

Листинг 1. Листинг программы

---

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <mpi.h>
4 #include <sys/time.h>
5
6 #define GNUPLOT
7 #define R 5 // Сопротивление
8 #define C 1 // Емкость
9 #define UN 0 // Начальные условия
10 #define UG 100 // Граничные условия
11 #define ht 1 // Шаг по времени
12 FILE* out, * fp; // Файл с данными для визуализации и файл gnuplot скрипта для
    отрисовки
13
14
15 void arg_abort(const char* msg) // Завершение программы при ошибке
16 {
17     printf("%s\n", msg);
18     MPI_Abort(MPI_COMM_WORLD, 0);
19     exit(0);
20 }
21
22 void gen_gnuplot(int time_interval, int element_c) // Генерация файла-скрипта для
    отображения графики
23 {
24     fprintf(fp, "set cbrange [0.9:1]\n");
25     fprintf(fp, "set xrange [0:%d]\n", element_c + 1);
26     fprintf(fp, "set yrange [-10:110]\n");
27     fprintf(fp, "set palette defined (1 '#ce4c7d')\n");
28     fprintf(fp, "set style line 1 lc rgb '#b90046' lt 1 lw 0.5\n");
29     fprintf(fp, "do for [i=0:%d]{\n", time_interval);
30     fprintf(fp, "plot 'out.txt' index i using 1:2 smooth bezier title 'U(t)'\npause 0.1}\npause
        -1\n");
31 }
32
33 void to_file(double* U, int element_c, double t) // Вывод в файл значений напряжений
34 {
35     int i, k;
36     for (i = 0; i < element_c + 2; i++)
37         fprintf(out, "%d\t%f\n", i, U[i]);
38     fprintf(out, "\n");
```

```

39  fprintf(out, "\\n\\n");
40  }
41
42  void begin_values(double *U_old_all, double *U_first, double *U_last, int total, long
    long element_c) // Задание начальных и граничных
    условий
43  {
44      for (long long i = 1; i <= element_c; i++)
45          U_old_all[i] = UN;
46      U_old_all[0] = UG;
47      U_old_all[element_c + 1] = UG;
48      for (int i = 1; i < total - 1; i++)
49      {
50          U_first[i] = UN;
51          U_last[i] = UN;
52      }
53      U_first[0] = UG;
54      U_last[total] = UG;
55  }
56
57  void calculate(double* U_new, double* U_old, double *first, double* last, long long
    n_per_proc) // Вычисление напряжений на новом
    шаге
58  {
59      for (long long i = 0; i < n_per_proc; i++)
60      {
61          if (i == 0)
62              U_new[i] = (first[0] - 2 * U_old[i] + U_old[i + 1]) * ht / R / C + U_old[i];
63          else if (i == n_per_proc - 1)
64              U_new[i] = (U_old[i - 1] - 2 * U_old[i] + last[0]) * ht / R / C + U_old[i];
65          else
66              U_new[i] = (U_old[i - 1] - 2 * U_old[i] + U_old[i + 1]) * ht / R / C + U_old[i];
67      }
68  }
69
70  int main(int argc, char* argv[])
71  {
72      double sum_sec = 0.0; // Переменная, запоминающая время, потраченное на отрисовку
73      int udif; // Переменная для вычисления разностей времен в микросекундах
74      double* U_old; // Значения напряжений в узлах на одном отрезке на предыдущем шаге
75      double* U_new; // Значения напряжений в узлах на одном отрезке на новом шаге
76      double* U_old_all; // Значения напряжений во всех узлах
77      double* U_first; // Значения напряжений в узлах прямо перед началами отрезков
78      double* U_last; // Значения напряжений в узлах прямо после концов отрезков
79      double first[1]; // Значение напряжения в узле перед началом отрезка
80      double last[1]; // Значение напряжения в узле после конца отрезка
81      long long n_per_proc; // Количество узлов на процесс (длина отрезка)

```

```

82 long long intBuf[2]; // Для передачи данных из аргументов вызова
83 int total, myrank; // Количество процессов и идентификатор конкретного процесса
84
85 timeval tv_s, tv1_e, tv2_e; // Время начала, конца выводов в файл и конца
    // выполнения программы
86 timezone tz; // Временная зона
87
88 MPI_Init(&argc, &argv); // Инициализация MPI
89 MPI_Comm_size(MPI_COMM_WORLD, &total); // Получение количества процессов
90 MPI_Comm_rank(MPI_COMM_WORLD, &myrank); // Получение идентификатора
    // процесса
91
92 if (argc < 3) // Проверка на количество аргументов запуска
93 {
94     if (!myrank)
95         arg_abort("Insert amount of elements that is divisible by 8 and time interval.");
96 }
97 #ifdef GNUPLOT
98 // Открытие файлов, нужных для визуализации
99 out = fopen("out.txt", "w");
100 fp = fopen("script.dat", "w");
101 if (fp == NULL || out == NULL) // Проверка на успешное открытие
102 {
103     arg_abort("Can't open file for writing.");
104 }
105 #endif
106 long long element_c = atoi(argv[1]); // Получение количества узлов из аргументов
107 long long time_interval = atoi(argv[2]); // Получение временного промежутка из
    // аргументов
108 if (element_c <= 0 || element_c % 8 != 0 || time_interval <= 0) // Проверка на
    // соответствие аргументов
109 {
110     if (!myrank)
111         arg_abort("Insert amount of elements that is divisible by 8 and time interval.");
112 }
113
114 #ifdef GNUPLOT
115 if (!myrank)
116     gen_gnuplot(time_interval, element_c); // Генерация скрипта для визуализации
117 #endif
118
119 if (!myrank) {
120     n_per_proc = element_c / total;
121     intBuf[0] = time_interval;
122     intBuf[1] = n_per_proc;
123 };
124 MPI_Bcast((void*)intBuf, 2, MPI_LONG_LONG, 0, MPI_COMM_WORLD); //
    // Рассылка данных, полученных в аргументах вызова

```

```

125 time_interval = intBuf[0];
126 n_per_proc = intBuf[1];
127 // Выделение памяти
128 U_new = (double*)malloc(sizeof(double) * n_per_proc);
129 U_old = (double*)malloc(sizeof(double) * n_per_proc);
130 U_old_all = (double*)malloc(sizeof(double) * (element_c + 2));
131 U_first = (double*)malloc(sizeof(double) * (total + 1));
132 U_last = (double*)malloc(sizeof(double) * (total + 1));
133 if (!myrank)
134 {
135     begin_values(U_old_all, U_first, U_last, total, element_c); // Определение начальных
        значений
136 }
137
138 if (!myrank)
139 {
140 #ifdef GNUPLOT
141     to_file(U_old_all, element_c, 0);
142 #endif
143     gettimeofday(&tv_s, &tz);
144 }
145 // Рассылка начальных значений в отрезках
146 MPI_Scatter((void*)(U_old_all + 1), n_per_proc, MPI_DOUBLE, (void*)U_old,
        n_per_proc, MPI_DOUBLE, 0, MPI_COMM_WORLD);
147 for (long long t = 1; t <= time_interval; t += ht)
148 {
149     // Рассылка значений, идущих прямо перед отрезками
150     MPI_Scatter((void*)U_first, 1, MPI_DOUBLE, (void*)first, 1, MPI_DOUBLE, 0,
        MPI_COMM_WORLD);
151     // Рассылка значений, идущих прямо после отрезков
152     MPI_Scatter((void*)(U_last + 1), 1, MPI_DOUBLE, (void*)last, 1, MPI_DOUBLE, 0,
        MPI_COMM_WORLD);
153     calculate(U_new, U_old, first, last, n_per_proc); // Вычисление значений на новом
        шаге
154     for (long long i = 0; i < n_per_proc; i++) // Теперь значения на новом шаге -
        значения на предыдущем шаге
155         U_old[i] = U_new[i];
156     // Подготовка для отправки новых предельных значений для других отрезков
157     last[0] = U_new[n_per_proc - 1];
158     first[0] = U_new[0];
159 #ifdef GNUPLOT
160     // Сбор всех полученных значений для вывода в файл
161     MPI_Gather((void*)U_new, n_per_proc, MPI_DOUBLE, (void*)(U_old_all + 1),
        n_per_proc, MPI_DOUBLE, 0, MPI_COMM_WORLD);
162     if (!myrank)
163     {
164         // Подсчет времени на вывод

```

```

165     gettimeofday(&tv1_e, &tz);
166     to_file(U_old_all, element_c, t);
167     gettimeofday(&tv2_e, &tz);
168     udif = tv2_e.tv_usec - tv1_e.tv_usec;
169     sum_sec += tv2_e.tv_sec - tv1_e.tv_sec - (udif < 0) + (udif + (udif < 0) *
        1000000) / 1000000.0;
170 }
171 #endif
172 // Сбор новых предельных значений
173 MPI_Gather((void*)last, 1, MPI_DOUBLE, (void*)(U_first + 1), 1, MPI_DOUBLE, 0,
        MPI_COMM_WORLD);
174 MPI_Gather((void*)first, 1, MPI_DOUBLE, (void*)(U_last), 1, MPI_DOUBLE, 0,
        MPI_COMM_WORLD);
175 }
176 if (!myrank)
177 {
178     // Вычисление времени работы программы с учетом выводов в файл
179     gettimeofday(&tv1_e, &tz);
180     udif = tv1_e.tv_usec - tv_s.tv_usec;
181     printf("Processing time: %lf s.\n", tv1_e.tv_sec - tv_s.tv_sec - (udif < 0) + (udif +
        (udif < 0) * 1000000) / 1000000.0 - sum_sec);
182 }
183 #ifdef GNUPLOT
184 // Заккрытие файлов для визуализации
185 fclose(out);
186 fclose(fp);
187 #endif
188 // Очистка памяти
189 free(U_old_all);
190 free(U_old);
191 free(U_new);
192 free(U_last);
193 free(U_first);
194 MPI_Finalize(); // Завершение работы с MPI
195 exit(0);
196 }

```

---