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

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине: «Разработка программных систем»

Студент	Харитонов Евгений Юрьевич
Группа	РК6-62б
Тип задания	лабораторная работа
Тема лабораторной работы	программирование средствами MPI

Студент _____ **Харитонов Е.Ю.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В.Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2020 г.

Оглавление

Задание на лабораторную работу

Вариант 3.

Разработать средствами MPI параллельную программу решения двухмерной нестационарной краевой задачи методом конечных разностей с использованием явной вычислительной схемы. Объект моделирования - прямоугольная пластина постоянной толщины. Возможны граничные условия первого и второго рода в различных узлах расчетной сетки. Временной интервал моделирования и количество (кратное 8) узлов по осям x и y расчетной сетки - параметры программы. Программа должна демонстрировать ускорение по сравнению с последовательным вариантом. Предусмотреть визуализацию результатов посредством утилиты gnuplot.

Описание структуры программы и реализованных способов взаимодействия процессов

При помощи средств MPI была разработана программа, выполняющаяся параллельно в рамках нескольких процессов.

В начале производится открытие файла для записи результатов. Если файл удалось открыть для чтения и записи, то с помощью функции MPI_Init происходит инициализация коммуникационных средств MPI. Далее определяется общее число параллельных процессов в группе MPI_COMM_WORLD при помощи функции MPI_Comm_size и записывается в переменную total. Далее каждый процесс записывает в переменную myrank свой номер в группе при помощи функции MPI_Comm_rank. Далее происходит проверка кратности длины M матрицы на количество процессов. Если количество не кратно, программа завершает работу.

Процесс root выделяет память под одномерный массив matrix размером M*N, где M - длина пластины, а N - ширина, а также инициализирует все элементы массива начальными значениями (начальными значениями в узлах пластины).

Каждый процесс будет рассчитывать свою часть пластины. Для этого пластина “режется” на total горизонтальных частей (рис. 1). Для расчета значений в узлах используется формула:

$$T_{i,j}^{t+1} = \frac{T_{i+1,j}^{t} - 2 \cdot T_{i,j}^{t} + T_{i-1,j}^{t}}{\Delta x^2} \cdot \Delta t + \frac{T_{i,j+1}^{t} - 2 \cdot T_{i,j}^{t} + T_{i,j-1}^{t}}{\Delta y^2} \cdot \Delta t + \cdot T_{i,j}^{t}$$

По формуле для расчета явной схемой видно, что для расчета понадобятся значения в соседних узлах, поэтому процессам придется обмениваться этими значениями. Для этих значений заведем массивы top_line и bottom_line. Поэтому в каждом процессе выделяем память под массивы:

prev - одномерный массив значений в узлах в предыдущий момент времени размером (M*N/total)

curr - одномерный массив значений в узлах в текущий момент времени размером (M*N/total)

top_line - одномерный массив значений в граничных узлах процесса myrank-1 в предыдущий момент времени размером N

bottom_line - одномерный массив значений в граничных узлах процесса myrank+1 в предыдущий момент времени размером N

Принцип деления узлов между процессами и расположение узлов, которые хранят массивы top_line и bottom_line, представлено на рис. 1

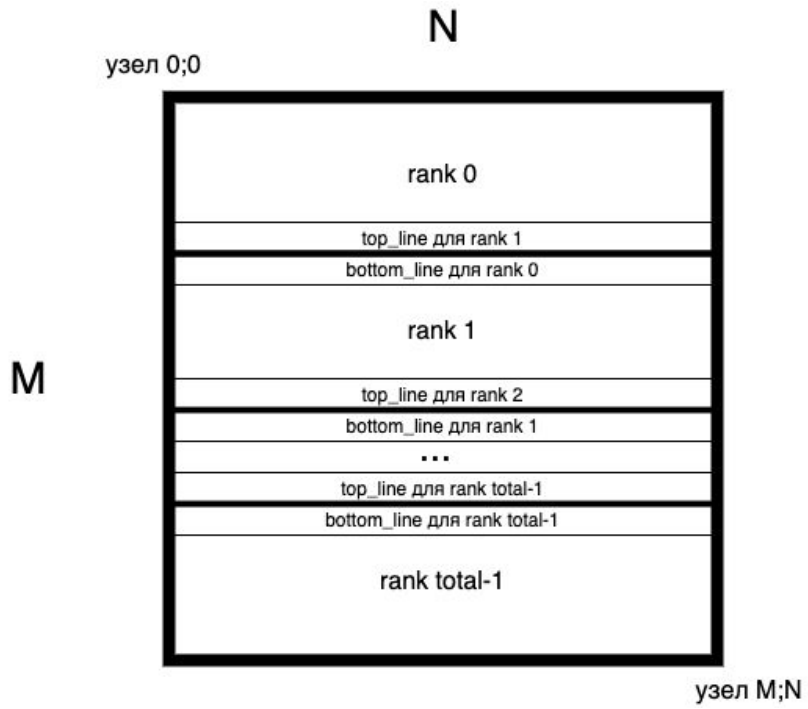


Рис. 1. Расположение частей пластины, которые считает каждый поток.

Для обмена актуальными значениями для top_line и bottom_line была написана функция send_line, в которой каждый процесс отправляет новые значения для top_line и bottom_line для соответствующих процессов, исключая крайние процессы. Функция send_line использует MPI_Send и MPI_Recv для обмена сообщениями между процессами.

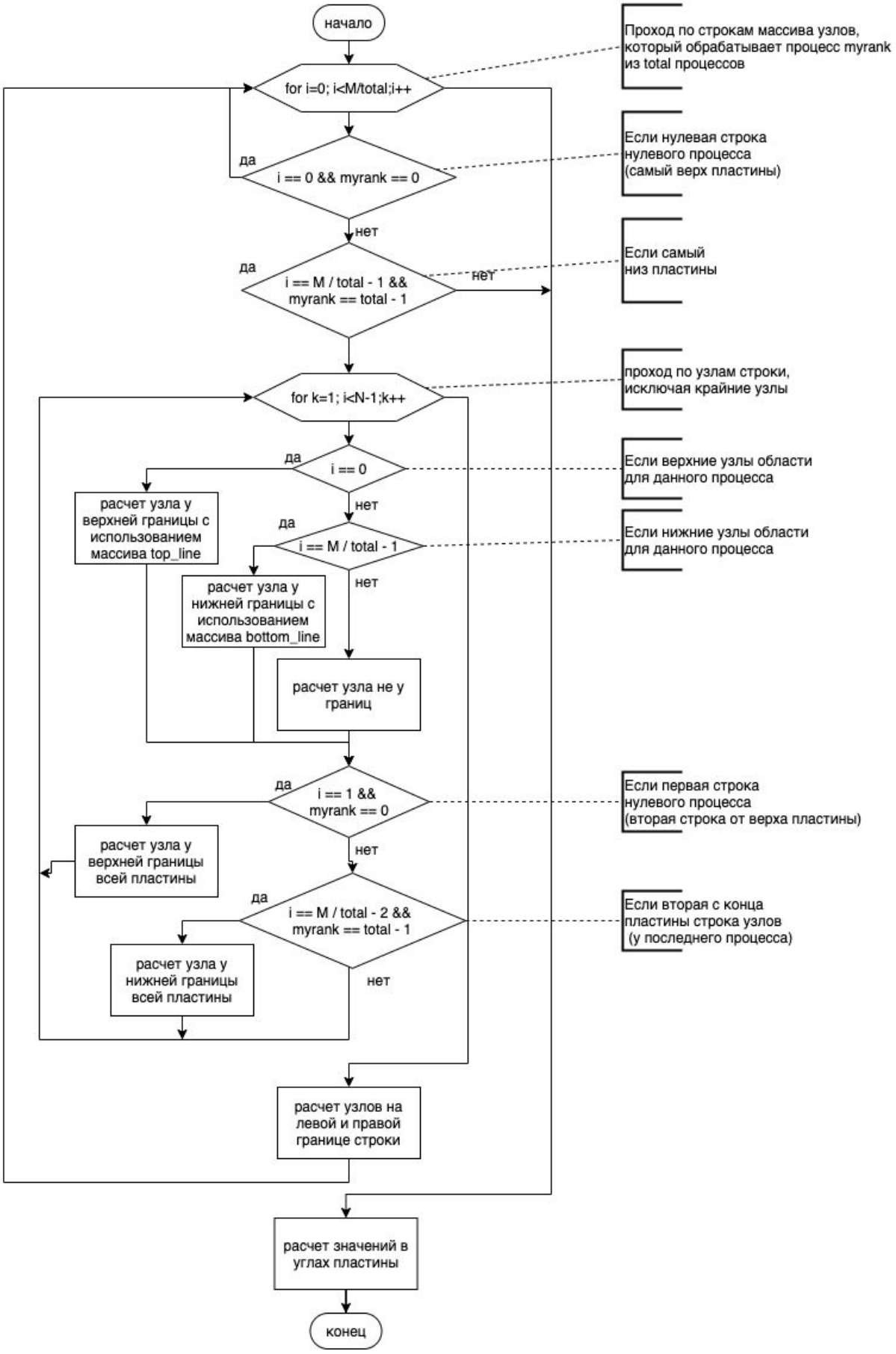
Далее каждый процесс инициализирует массив prev начальными значениями и начинается работа расчетного цикла. Цикл TIME раз вызывает функцию send_line, запускает функцию расчета значений в узлах, которые рассчитывает конкретный процесс и вызывает функцию MPI_Gather, с помощью которой процесс root получает новые значения в узлах от других процессов и записывает значения в файл.

После расчета значений во всех временных слоях все процессы, кроме root, завершают свою работу, а процесс root выводит полученные значения из файла на экран с помощью gnuplot.

Блок-схема


начало
открытие файла для результов
MPI_Init
MPI_Comm_size
MPI_Comm_rank
если процесс root
нет
да
создание матрицы matrix для всех узлов пластины
создание матриц для узлов каждого процесса в текущий и предыдущий момент времени
создание матриц для граничных узлов
инициализация матриц начальными значениями
for(i = 0; i < TIME; i++)
Обмен граничными значениями между процессами
расчет значений в узлах
Сбор процессом root значений во всех узлах пластины с помощью MPI_Garhet
если процесс root
нет
да
запись в файл
Указатели на массивы значений в предыдущий и текущий момент времени меняются местами
если процесс root
нет
да
вывод графика с помощью gnuplot
конец

Функция расчета узлов



Результаты работы программы

Результат работы программы при M=32, N=16 на временном слое 100 представлен на рис. 2

Граничные условия при условии, что узел 0;0 расположен в верхнем левом углу пластины:

Верхняя граница: ГУ первого рода, температура 20 градусов

Нижняя граница: ГУ второго рода, теплоизолирована

Левая граница: ГУ первого рода, температура 70 градусов

Правая граница: ГУ второго рода, теплоизолирована

Узел M;N (противоположный 0;0): ГУ второго рода

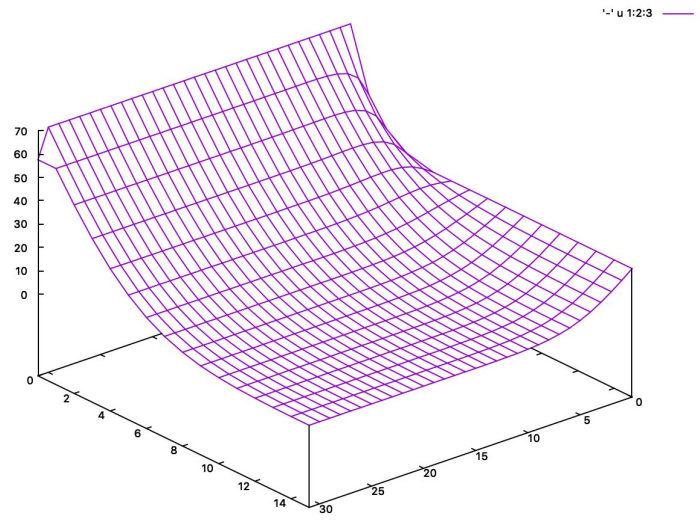


Рис. 2. Результат работы программы.

Для оценки эффективности разбиения на процессы были сделаны замеры времени работы с помощью утилиты time при разном числе процессов на сетке размером 4096 на 2048 и временем работы 1000. Результаты представлены в таблице 1.

Количество процессов	real	user	sys
1	3m5,301s	3m5,243s	0m0,127s
2	1m41,439s	3m22,754s	0m0,274s
4	0m55,834s	3m40,259s	0m0,543s
8	0m32,919s	4m2,638s	0m1,159s

Таблица 1. Время работы программы при разном количестве процессов

Текст программы

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <mpi.h>

#define LEFT_EDGE_TYPE 1
#define LEFT_EDGE_VALUE 70

#define RIGHT_EDGE_TYPE 2
#define RIGHT_EDGE_VALUE 0

#define TOP_EDGE_TYPE 1
#define TOP_EDGE_VALUE 20

#define BOTTOM_EDGE_TYPE 2
#define BOTTOM_EDGE_VALUE 0

#define POINT_VALUE 5

#define START_T 0   //стартовая температура пластины
#define dx 1
#define dy 1
#define dt 0.1
size_t M;              //длина
size_t N;              //ширина
size_t TIME;           //время расчета

#define GRAPH 0


void write_res_to_file(FILE *f, double *arr) {
   for(size_t i = 0; i < M; i++) {
      for(size_t k = 0; k < N; k++) {
         fprintf(f, "%zu %zu %lf\n", i, k, arr[i * N + k]);
      }
   }
   fprintf(f, "\n");
}

void send_line(double *prev, double *top_line, double *bottom_line, int myrank, int total) {
   if(myrank != total - 1) {
      MPI_Send(prev + (M / total - 1) * N, N, MPI_DOUBLE, myrank + 1, 0,
MPI_COMM_WORLD);
   }
   if(myrank != 0) {
```

```
      MPI_Recv(top_line, N, MPI_DOUBLE, myrank - 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      MPI_Send(prev, N, MPI_DOUBLE, myrank - 1, 0, MPI_COMM_WORLD);
   }
   if(myrank != total - 1) {
      MPI_Recv(bottom_line, N, MPI_DOUBLE, myrank + 1, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
   }
}

int solver(double *prev, double *curr, double *top_line, double *bottom_line, int myrank, int total) {
   for(size_t i = 0; i < M / total; i++) {
      if(i == 0 && myrank == 0) {
         continue;
      }
      if (i == M / total - 1 && myrank == total - 1) {
         break;
      }

      for(size_t k = 1; k < N - 1; k++) {
         if (i == 0) {
            curr[i * N + k] = prev[i * N + k] + dt * ((prev[i * N + k - 1] + prev[i * N + k + 1] - 2 *
prev[i * N + k]) / (dx * dx) + (prev[(i + 1) * N + k] + top_line[k] - 2 * prev[i * N + k]) / (dy * dy));
         } else if (i == M / total - 1) {
            curr[i * N + k] = prev[i * N + k] + dt * ((prev[i * N + k - 1] + prev[i * N + k + 1] - 2 *
prev[i * N + k]) / (dx * dx) + (bottom_line[k] + prev[(i - 1) * N + k] - 2 * prev[i * N + k]) / (dy * dy));
         } else {
            curr[i * N + k] = prev[i * N + k] + dt *
                                    ((prev[i * N + k - 1] + prev[i * N + k + 1] - 2 * prev[i * N + k]) /
                                     (dx * dx) + (prev[(i + 1) * N + k] + prev[(i - 1) * N + k] -
                                             2 * prev[i * N + k]) / (dy * dy));
         }
         if (i == 1 && myrank == 0) {
            if (TOP_EDGE_TYPE == 1) {
               curr[k] = TOP_EDGE_VALUE;
            } else if (TOP_EDGE_TYPE == 2) {
               curr[k] = curr[(i) * N + k] - dx * TOP_EDGE_VALUE;
            }
         }
         if (i == M / total - 2 && myrank == total - 1) {
            if (BOTTOM_EDGE_TYPE == 1) {
               curr[(i + 1) * N + k] = BOTTOM_EDGE_VALUE;
            } else if (BOTTOM_EDGE_TYPE == 2) {
               curr[(i + 1) * N + k] = curr[(i) * N + k] - dy * BOTTOM_EDGE_VALUE;
            }
         }
      }
      if (LEFT_EDGE_TYPE == 1) {
         curr[i * N] = LEFT_EDGE_VALUE;
      } else if (LEFT_EDGE_TYPE == 2) {
```

```
            curr[i * N] = curr[i * N + 1] - dy * LEFT_EDGE_VALUE;
        }

        if (RIGHT_EDGE_TYPE == 1) {
            curr[(i + 1) * N - 1] = RIGHT_EDGE_VALUE;
        } else if (RIGHT_EDGE_TYPE == 2) {
            curr[(i + 1) * N - 1] = curr[(i + 1) * N - 2] - dy * RIGHT_EDGE_VALUE;
        }
    }

    //углы
    if (myrank == 0) {
        curr[0] = curr[1];
        curr[N - 1] = curr[N - 2];
    }
    if (myrank == total - 1) {
        curr[(M / total - 1) * N] = curr[(M / total - 1) * N + 1];
        curr[(M / total) * N - 1] = (curr[(M / total) * N - 2] * dx + curr[(M / total - 1) * N - 1] * dy +
POINT_VALUE * dx * dy) / (dx + dy);
    }
    return 0;
}

int main(int argc, char **argv) {

    FILE *result_file = fopen("result", "w+");
    if (result_file < 0) {
        exit(EXIT_FAILURE);
    }

    int myrank;
    int total;
    MPI_Init (&argc, &argv);
    MPI_Comm_size (MPI_COMM_WORLD, &total);
    MPI_Comm_rank (MPI_COMM_WORLD, &myrank);

    TIME = atoll(argv[1]);
    M = atoll(argv[2]);
    N = atoll(argv[3]);
    if (M % total != 0) {
        if (!myrank) {
            printf("Wrong length");
        }
        MPI_Finalize();
        exit(EXIT_FAILURE);
    }

    int len = M / total;
    double *matrix = NULL;
    if(!myrank) {
```

```
        matrix = malloc(M * N * sizeof(double));
        printf("Calculation...\n");
    }

    double *curr = malloc(N * len * sizeof(double));
    double *prev = malloc(N * len * sizeof(double));
    double *top_line = malloc(N * sizeof(double));
    double *bottom_line = malloc(N * sizeof(double));

    for(size_t i = 0; i < len; i++) {
        for(size_t k = 0; k < N; k++) {
            prev[i * N + k] = START_T;
        }
    }

    for (size_t i = 0; i < TIME; i++) {
        send_line(prev, top_line, bottom_line, myrank, total);
        solver(prev, curr, top_line, bottom_line, myrank, total);
        MPI_Gather(curr, len * N, MPI_DOUBLE, matrix, len * N, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
        double *tmp = prev;
        prev = curr;
        curr = tmp;
        if(GRAPH) {
            if (!myrank) {
                write_res_to_file(result_file, matrix);
            }
        }
    }
    free(matrix);
    free(curr);
    free(prev);
    free(top_line);
    free(bottom_line);
    if(myrank) {
        MPI_Finalize();
        exit(0);
    }
    printf("Calculated!\n");

    if (!GRAPH) {
        write_res_to_file(result_file, matrix);
        fclose(result_file);
        MPI_Finalize();
        return 0;
    }

    size_t m = 0;
    size_t n = 0;
    double d = 0;
```

```
    fseek(result_file, 0, SEEK_SET);

    FILE *gnuplot = popen("gnuplot -persist", "w");     //графики
    if (gnuplot == NULL) {
        printf("gnuplot error\n");
        exit(EXIT_FAILURE);
    }

    printf("%zu %zu\n", M, N);
    if (!myrank) {
        fprintf(gnuplot, "set dgrid3d %zu, %zu \n", N, M);
        fprintf(gnuplot, "set hidden3d\n");
        fprintf(gnuplot,"set xrange[0:%zu]\nset yrange[0:%zu]\n", M- 1, N - 1);
        for(int i = 0; i < TIME;i++) {
            fprintf(gnuplot, "splot '-' u 1:2:3 with lines\n");
            for(size_t i = 0; i < M * N; i++) {
                fscanf(result_file, "%zu %zu %lf", &m, &n, &d);
                fprintf(gnuplot, "%zu %zu %lf\n", m, n, d);
            }
            fprintf(gnuplot, "e\n");
            fflush(gnuplot);
            fprintf(gnuplot,"pause(0.1)\n");
        }
    }

    fclose(result_file);
    fclose(gnuplot);
    MPI_Finalize();
    return 0;
}
```