Министерство образования и науки 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 №4

Студент	Журавлев Николай Вадимович
Группа	РК6-62б
Тип задания	Лабораторная работа
Тема лабораторной работы	Программирование средствами MPI

Студент _____ **Н.В. Журавлев**
подпись, дата *фамилия, и.о.*

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

Оценка _____

Москва, 2022 г.

Оглавление

Текст задания

Разработать средствами MPI параллельную программу решения двухмерной нестационарной краевой задачи методом конечных разностей с использованием явной вычислительной схемы. Объект моделирования - прямоугольная пластина постоянной толщины. Подробности постановки подобной задачи даны ниже. 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. При этом утилита gnuplot должна вызываться отдельной командой после окончания расчета.

Описание структуры программы и реализованных способов взаимодействия процессов

При помощи средств MPI была разработана программа, которая выполняется параллельно в рамках нескольких процессов, функционирующих одновременно.

С помощью функции MPI_Init происходит инициализация коммуникационных средств MPI. После этого необходимо определить общее количество параллельных процессов в группе с помощью функции MPI_Comm_size. Функция MPI_Comm_rank нужна для определения номера процесса в группе.

Для процесса с номером 0 выделяется память под одномерный массив A1 размером N*M, где N и M – ширина и высота пластины соответственно. После этого вызывается функция init_matrix. Она заполняет массив в соответствии с начальным состоянием пластины.

Каждый процесс будет считать свою часть пластины. Пластина делится на полоски шириной n и высотой M/total, где total – количество процессов. При помощи функции MPI_Bcast процесс с идентификатором 0 рассылает всем процессам в группе сообщение из буфера. В буфере хранятся данные о размере

рассчитываемых частей, а также о времени расчета. Все процессы (в том числе и процесс 0) в группе принимают сообщение в буфер.

В каждом процессе выделяется память для двух одномерных массивов a0 и a1 размером n*m, где n и m – ширина и высота рассчитываемой части соответственно.

Так же выделяется память для двух одномерных массивов a_neighbour_u и a_neighbour_down размером n. В них будут храниться значения на границах соседних частей. Это необходимо для расчета граничных значений текущей части.

Для одновременной рассылки разных (но однотипных) данных разным процессам в группе используется функция MPI_Scatter. При помощи нее процесс с индентификатором 0 распределяет по всем процессам в группе содержимое буфера. Таким образом, каждый процесс будет иметь данные о своей части. Эти данные будут храниться в созданном ранее массиве a0 размером n*m.

Для каждого момента времени необходимо рассчитать значения в узлах пластины. С помощью функции send_line заполняются массивы a_neighbour_up и a_neighbour_down. В ней используется функция MPI_Sendrecv для обмена данными между процессами. Схема обмена информации представлена на рис.1.

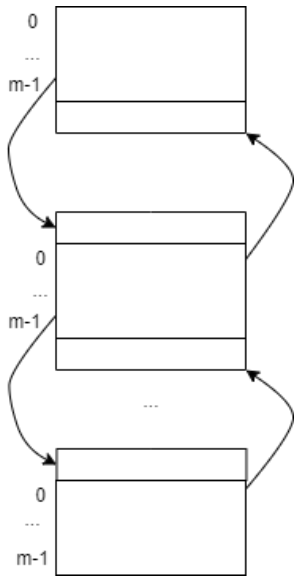

Рисунок 1. Схема передачи информации о соседних узлах

Потом при помощи функции solve рассчитываются значения в узлах по формулам для явной вычислительной схемы и заносятся в массив a1. После этого собираются данные с процессов при помощи MPI_Gather. Процесс 0 принимает данные со всех процессов и заносит в одномерный массив A1 со всеми значениями.

После этого, если это родительский процесс, то данные из массива, хранящего все значения в узлах, заносятся в файл результатов. После окончания всех вычислений завершение обменов осуществляется функцией MPI_Finalize.

Были созданы два файла. В одном из них хранятся значения в узлах пластин, в другом файле хранится информация, необходимая для вывода графиков на экран. Файл с значениями заполняется каждый раз, когда рассчитан новый временной слой. Файл для вывода графиков на экран заполняется один раз процессом с идентификатором 0.

Описание основных используемых структур данных

Создан одномерный массив A1 в родительском процессе размером m*n для хранения информации о всей пластине. В каждом из процессов созданы два одномерных массива a0 и a1 размером m*n/total. В них хранятся значения для данной части пластины в текущий момент времени и в прошлый момент времени. Для расчета узлов на границах созданы два одномерных массива a_neighbour_up и a_neighbour_down размером n. В них хранятся значения в узлах на границах соседних частей пластины. m - высота пластины, n - ширина пластины, total - количество процессов.

Блок-схема программы

Блок-схема программы на рисунках 2, 3, 4.

Рисунок 2. Инициализация программы

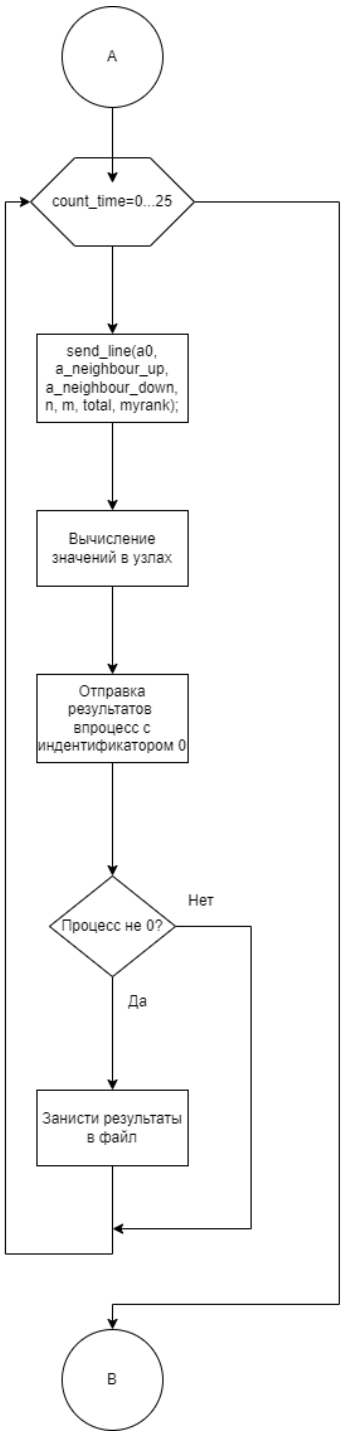

Рисунок 3. Основной цикл программы

Рисунок 4. Завершение программы

Блок-схема функции send_line представлена на рис.4.

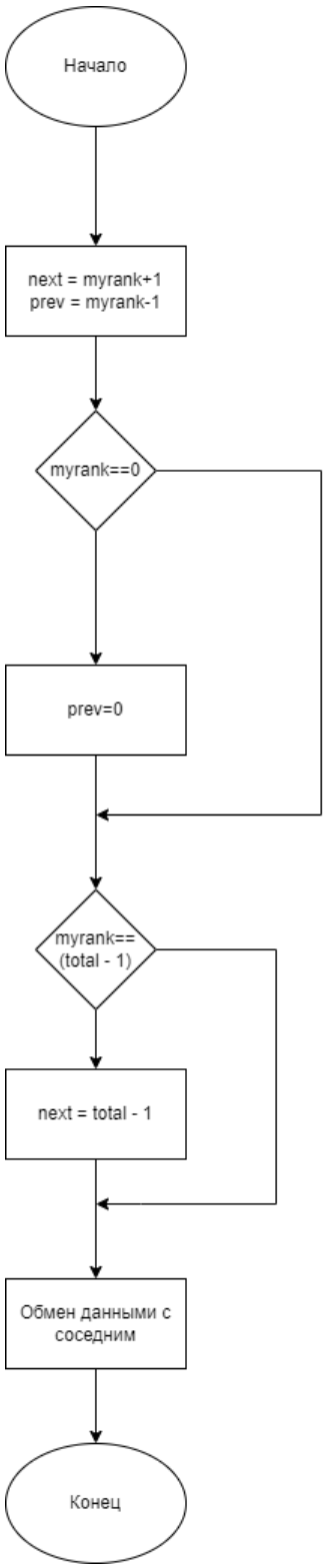

Рисунок 5. Блок схема функции send_line

Примеры результатов работы программы

При наложении граничных условий 2 рода на левую часть и первого на ближнюю получится график на рис.6.

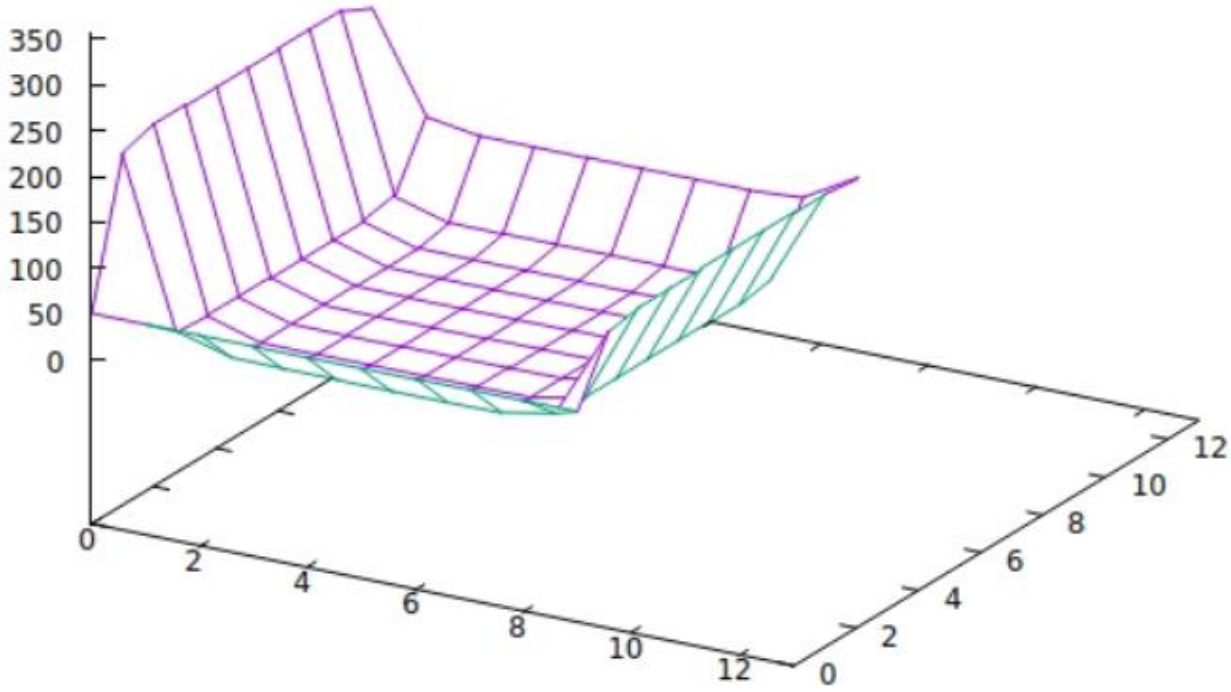

Рисунок 6. График для данных

Были получены временные затраты на расчет. Для пластины 2048x2048 и времени в 2048 секунд (шаг равен 1 секунде) результаты представлены в таблице 1.

Таблица 1

Количество потоков	Время, ms
1	88 376
2	42 739
4	20 192
8	10 421

Текст программы

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi/mpi.h>
```

```
#include <sys/time.h>
#define GNUPLOT
#define AT 0.00008838
#define HT 1
#define HX 0.1
#define HY 0.1
// Начальные условия
#define LEFT_CONSTRAINT 100
#define RIGHT_CONSTRAINT 100
#define TOP_CONSTRAINT 100
#define BOTTOM_CONSTRAINT 100

// Граничные условия 1-ого рода
#define FIRST_CONDITION_LEFT_CONSTRAINT 0
#define FIRST_CONDITION_RIGHT_CONSTRAINT 0
#define FIRST_CONDITION_TOP_CONSTRAINT 0
#define FIRST_CONDITION_BOTTOM_CONSTRAINT 1

// Граничные условия 2-ого рода
#define SECOND_CONDITION_LEFT_CONSTRAINT 0
#define SECOND_CONDITION_RIGHT_CONSTRAINT 0
#define SECOND_CONDITION_TOP_CONSTRAINT 0
#define SECOND_CONDITION_BOTTOM_CONSTRAINT 0
void init_matrix(double* A1, int n, int m) {
    int i, j;
    //заполнение краевых значений
    for (i = 0; i < n; i++) {
        A1[i] = TOP_CONSTRAINT;
        A1[(n * (m - 1)) + i] = BOTTOM_CONSTRAINT;
    }

    for (i = 0, j = 0; i < m; i++, j += n) {
        A1[j] = LEFT_CONSTRAINT;
        A1[(j + n) - 1] = RIGHT_CONSTRAINT;
    }
```

```
    }
    void send_line(double* a0, double* a_neighbour_up, double* a_neighbour_down, int n, int m, int total, int myrank) {
        int next, prev;
        next = myrank + 1;
        prev = myrank - 1;
        if (myrank == 0) {
            prev = 0;
        }
        if (myrank == (total - 1)) {
            next = total - 1;
        }

        MPI_Sendrecv((void*)&a0[n * m - n], n, MPI_DOUBLE, next, 0,
                (void*)a_neighbour_down, n, MPI_DOUBLE, next, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Sendrecv((void*)&a0[0], n, MPI_DOUBLE, prev, 0, (void*)a_neighbour_up, n,
                MPI_DOUBLE, prev, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    void solve(double* a1, const double* a0, const double* a_neighbour_up, const double* a_neighbour_down, int n, int m, int myrank, int total) {
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (i == (m - 1)) {
                    if (myrank != (total - 1)) {
                        if (j == 0) {
                            a1[i * n + j] = AT * HT *
                                    ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + 0) / (HX * HX) +
                                     (a_neighbour_down[j] - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) +
                                    a0[i * n + j];
                        } else {
                            if (j == (n - 1)) {
                                a1[i * n + j] = AT * HT *
                                        ((0 - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX * HX) +
```

```
                                (a_neighbour_down[j] - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY
* HY)) +
                                a0[i * n + j];
                        } else {
                            a1[i * n + j] = AT * HT *
                                ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) /
                                (HX * HX) +
                                (a_neighbour_down[j] - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) /
                                (HY * HY)) +
                                a0[i * n + j];
                        }
                    }
                } else {
                    if (j == 0) {
                        a1[i * n + j] = AT * HT *
                            ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + 0) / (HX * HX) +
                            (0 - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) +
                            a0[i * n + j];
                    } else {
                        if (j == (n - 1)) {
                            a1[i * n + j] = AT * HT *
                                ((0- 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX * HX) +
                                (0 - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) +
                                a0[i * n + j];
                        } else {
                            a1[i * n + j] = AT * HT *
                                ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX *
HX) +
                                (0 - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) +
                                a0[i * n + j];
                        }
                    }
                }
                continue;
            }
```

```
                if (i == 0) {
                    if (myrank != 0) {
                        if (j == 0) {
                            a1[i * n + j] = AT * HT *
                                    ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + 0) / (HX * HX) +
                                     (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + a_neighbour_up[j]) / (HY *
HY)) + a0[i * n + j];
                        } else {
                            if (j == (n - 1)) {
                                a1[i * n + j] = AT * HT *
                                        ((0 - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX * HX) +
                                         (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + a_neighbour_up[j]) / (HY
* HY)) + a0[i * n + j];
                            } else {
                                a1[i * n + j] = AT * HT *
                                        ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX *
HX) +
                                         (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + a_neighbour_up[j]) / (HY
* HY)) +
                                        a0[i * n + j];
                            }
                        }
                    } else {
                        if (j == 0) {
                            a1[i * n + j] = AT * HT *
                                    ((0 - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX * HX) +
                                     (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + 0) / (HY * HY)) + a0[i * n +
j];
                        } else {
                            if (j == (n - 1)) {
                                a1[i * n + j] = AT * HT *
                                        ((0 - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX * HX) +
                                         (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + 0) / (HY * HY)) + a0[i * n
+ j];
                            } else {
```

```
                        a1[i * n + j] = AT * HT *
                                ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX *
HX) +
                                (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + 0) / (HY * HY)) + a0[i * n
+ j];
                    }
                  }
                }
                continue;
              }

              if (j == 0) {
                a1[i * n + j] = AT * HT *
                        ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + 0) / (HX * HX) +
                         (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) +
a0[i * n + j];
                continue;
              }
              if (j == (n - 1)) {
                a1[i * n + j] = AT * HT *
                     ((0 - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) / (HX * HX) +
                     (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) + a0[i
* n + j];
                continue;
              }
              a1[i * n + j] = AT * HT * ((a0[i * n + (j + 1)] - 2 * a0[i * n + j] + a0[i * n + (j - 1)]) /
(HX * HX)
                  + (a0[(i + 1) * n + j] - 2 * a0[i * n + j] + a0[(i - 1) * n + j]) / (HY * HY)) + a0[i *
n + j];
            }
          }
          if (myrank == 0) {
            for (int i = 0; i < n; i++) {
              a1[i] = (-1) * HY * SECOND_CONDITION_TOP_CONSTRAINT + a1[i];
            }
```

```
        }

        if ((total - 1) == myrank) {
            for (int j = 0; j < n; j++) {
                a1[(m - 1) * n + j] = (-1) * HY * SECOND_CONDITION_BOTTOM_CONSTRAINT
+ a1[(m - 1) * n + j];
            }
        }
        for (int i = 0; i < m; i++) {
            a1[i * n] = (-1) * HX * SECOND_CONDITION_LEFT_CONSTRAINT + a1[i * n];
        }

        for (int i = 0; i < m; i++) {
            int pos = i * n + (n - 1);
            a1[pos] = (-1) * HX * SECOND_CONDITION_RIGHT_CONSTRAINT + a1[pos];
        }
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (myrank == 0 && i == 0 && FIRST_CONDITION_TOP_CONSTRAINT) {
                    a1[i * n + j] = TOP_CONSTRAINT;
                    continue;
                }
                if (myrank == (total - 1) && i == (m - 1) &&
FIRST_CONDITION_BOTTOM_CONSTRAINT) {
                    a1[i * n + j] = BOTTOM_CONSTRAINT;
                    continue;
                }
                if (j == 0 && FIRST_CONDITION_LEFT_CONSTRAINT) {
                    a1[i * n + j] = LEFT_CONSTRAINT;
                    continue;
                }
                if (j == (n - 1) && FIRST_CONDITION_RIGHT_CONSTRAINT) {
                    a1[i * n + j] = RIGHT_CONSTRAINT;
                    continue;
                }
```

```
        }
    }
}
void make_gnu(double* M, FILE* fds1, int n, int m) {
    int i, j, k;
    for (i = 0, k = m - 1; i < m; i++, k--) {
        for (j = 0; j < n; j++)
            fprintf(fds1, " %d %d %3lf\n", j, k, M[i * n + j]);
        fprintf(fds1, "\n");
    }
    fprintf(fds1, "\n\n");
}
void copy(double* a, const double* b, int n, int m) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            a[i * n + j] = b[i * n + j];
        }
    }
}
int main(int argc, char** argv) {printf("\n");
    int n, m, N, M, t, count_time;
    double *A1, *a0, *a1, *a_neighbour_up, *a_neighbour_down;
    int total, myrank;
    int intBuf[3];
    N = atoi(argv[1]);
    M = atoi(argv[2]);
    t = atoi(argv[3]);
    FILE *fds1, *fds2;
    fds1 = fopen("result.txt", "w");
    fds2 = fopen("file.gnu", "w");
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &total);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    if (!myrank) {
        intBuf[0] = N;
```

```
        intBuf[1] = M / total; //высота полосы обработки
        intBuf[2] = t;
        A1 = (double*)malloc(sizeof(double)*N*M);
        init_matrix(A1, N, M);
        make_gnu(A1, fds1, atoi(argv[1]), atoi(argv[2]));
    }

    MPI_Bcast((void*)intBuf, 3, MPI_INT, 0, MPI_COMM_WORLD);
    n = intBuf[0];
    m = intBuf[1];
    t = intBuf[2];

    a0 = (double*)malloc(sizeof(double) * n * m);
    a1 = (double*)malloc(sizeof(double) * n * m);
    a_neighbour_up = (double*)malloc(sizeof(double) * n);
    a_neighbour_down = (double*)malloc(sizeof(double) * n);
    MPI_Scatter((void*)A1, n * m, MPI_DOUBLE, (void*)a0, n * m, MPI_DOUBLE, 0,
            MPI_COMM_WORLD); // send A1 in a0 from root
    copy(a1, a0, n, m);
    struct timeval tv1, tv2, dtv;
    struct timezone tz;
    gettimeofday(&tv1, &tz);
    for (count_time = 0; count_time < t; count_time++) {
        send_line(a0, a_neighbour_up, a_neighbour_down, n, m, total, myrank);
        solve(a1, a0, a_neighbour_up, a_neighbour_down, n, m, myrank, total);
//        print_matrix(a1, n, m, myrank);
        MPI_Gather((void*)a1, n * m, MPI_DOUBLE, (void*)A1, n * m, MPI_DOUBLE, 0,
                MPI_COMM_WORLD);
//        if (myrank == 0) print_matrix(A1,n , m * total, 2);
        copy(a0, a1, n, m);
        if (!myrank) {
            make_gnu(A1, fds1, atoi(argv[1]), atoi(argv[2]));
        }
    }
    if (!myrank) {
```

```
//        print_matrix(A1, n, m * total,0);
            int max = 0;
            int min = 0;
            for (int i = 0; i < atoi(argv[2]); i++) {
               for (int j = 0; j < n; j++) {
                 if (A1[i * n + j] > max) {
                    max = (int) A1[i * n + j];
                 }
                 if (A1[i * n + j] < min) {
                    min = (int) A1[i * n + j];
                 }
               }
            }
            fprintf(fds2, "set dgrid3d\n");
            fprintf(fds2, "set hidden3d\n");
            fprintf(fds2,"set xrange[0:%d]\nset yrange[0:%d]\nset zrange[%d:%d]\n", atoi(argv[1])
+ 3, atoi(argv[2]) + 3, min - 1, max > 200 ? max + 1: 200);
            for (int i = 0; i < (int)(t/HT); i++) {
               fprintf(fds2,"%s","splot 'result.txt' ");
               fprintf(fds2,"index %d using 1:2:3 with lines\n", i);
               fprintf(fds2,"pause(0.1)\n");
            }
        #ifdef GNUPLOT
            system("gnuplot file.gnu -persist");
        #endif
          }
          gettimeofday(&tv2, &tz);
          dtv.tv_sec = tv2.tv_sec - tv1.tv_sec;
          dtv.tv_usec = tv2.tv_usec - tv1.tv_usec;

          if (dtv.tv_usec < 0) {
             dtv.tv_sec--;
             dtv.tv_usec += 1000000;
          } // костыль
          printf("%d s %ld ms\n", (int)dtv.tv_sec, dtv.tv_usec / 1000);
```

```
    MPI_Finalize();
    exit(0);
}
```