###### 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 ЛАБОРАТОРНОЙ РАБОТЫ**

«Умножение матрицы на матрицу в MPI 2D решетка»

студента 2 курса, 21203 группы

**Неретина Степана Ивановича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

А.Ю. Власенко

Новосибирск 2023

**СОДЕРЖАНИЕ**

[ЗАДАНИЕ 3](#_heading=h.gjdgxs)

[ОПИСАНИЕ РАБОТЫ 4](#_heading=h.30j0zll)

Вывод8

[Приложение 1 Исходный код программы](#_heading=h.1fob9te)  9

# ЗАДАНИЕ

1. Реализовать параллельный алгоритм умножения матрицы на матрицу

при 2D решетке процессов с соблюдением требований (см.выше).

1. Исследовать производительность параллельной программы при

фиксированном размере матрицы в зависимости от и размера решетки:

2x12, 3x8, 4x6, 6x4, 8x3, 12x2. Размер матриц подобрать таким образом,

чтобы худшее из времен данного набора было не менее 30 сек.

1. Выполнить профилирование программы при использовании 8-и ядер

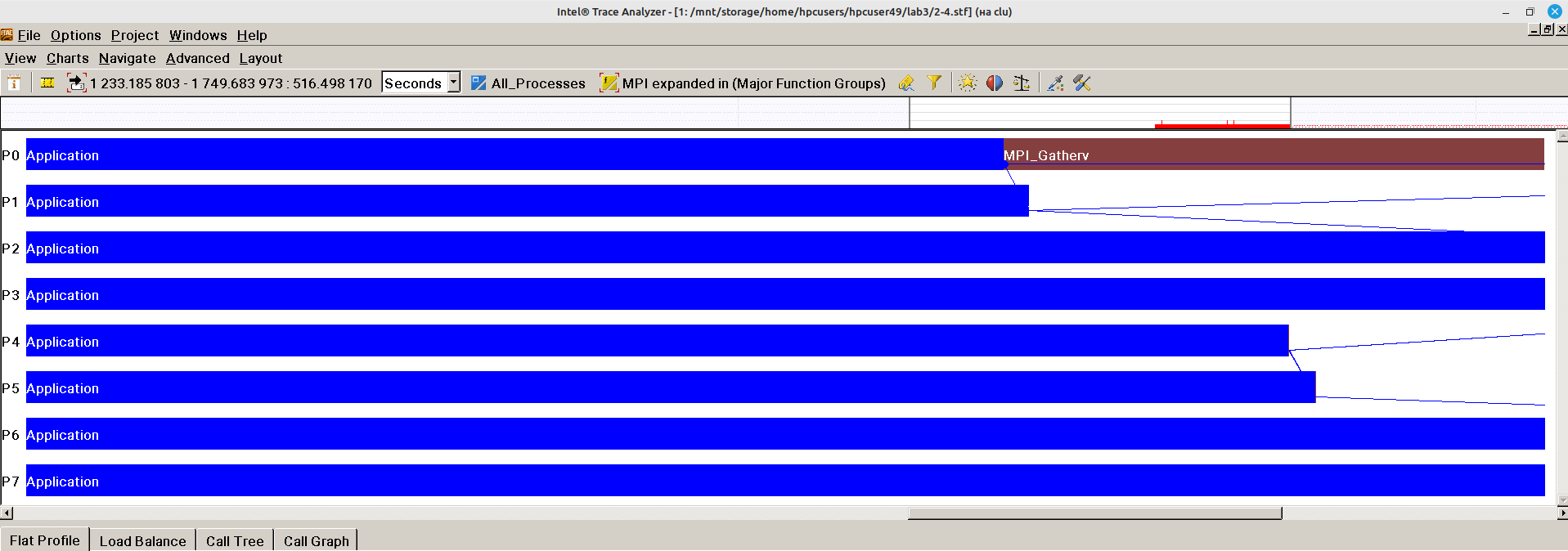
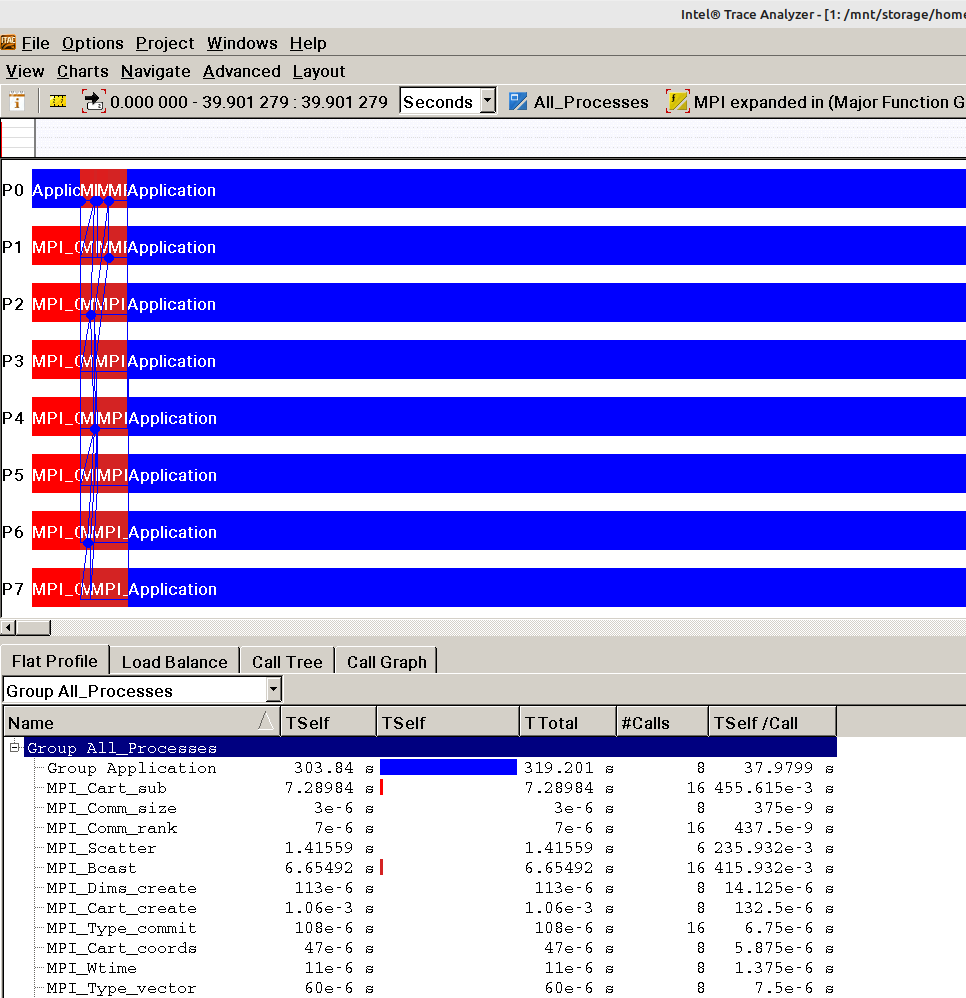
с решетками 2x4, 4x2.

# ОПИСАНИЕ РАБОТЫ

# Реализован алгоритм из задания, произведены замеры времени и профилирование на кластере

|  |  |
| --- | --- |
| Топология | Время(сек) |
| 12-2 | 787.721085 |
| 2-12 | 698.799692 |
| 3-8 | 807.286655 |
| 4-6 | 688.516149 |
| 6-4 | 797.157500 |
| 8-3 | 819.660823 |

**Профилирование 2-4**

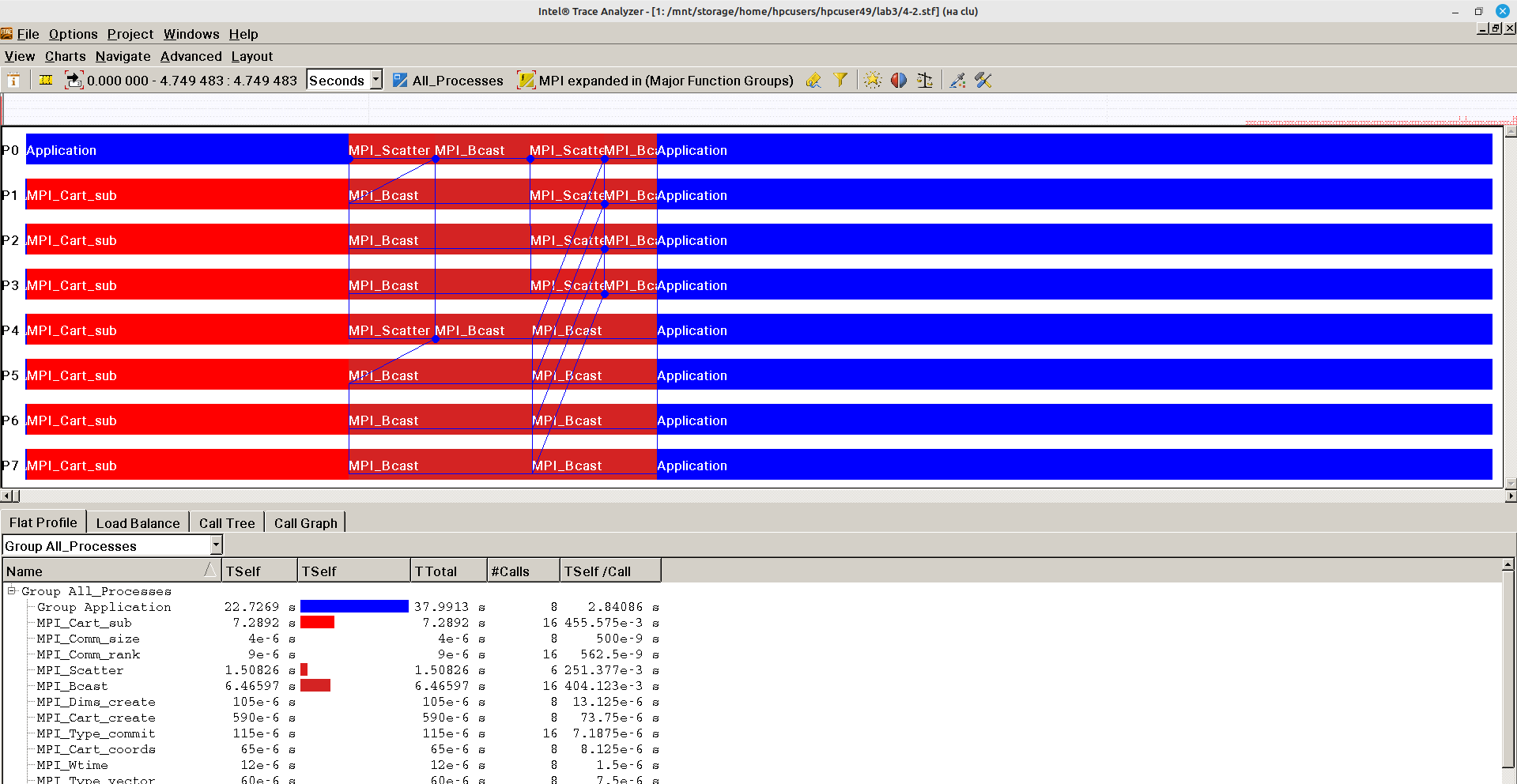


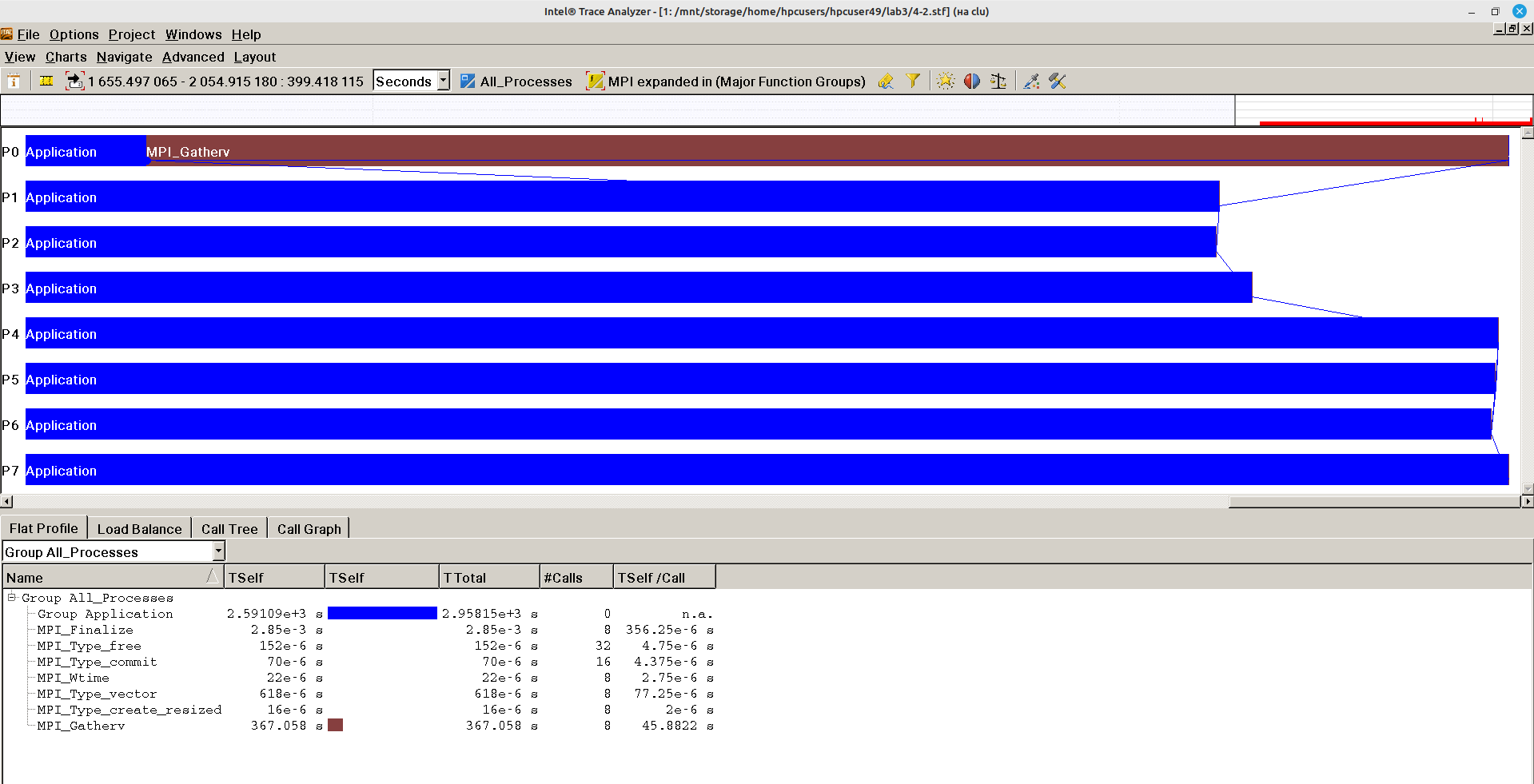
# 

# 

# 

**Профилирование 4-2**





# **Вывод**

В ходе выполнения данной лабораторной работы мной были освоены концепции коммуникаторов и декартовых топологий. Приобретены навыки работы с производными типами и методы работы с ними при помощи MPI функций.

# **Приложение 1. *Исходный код программы***

**#include <stdio.h>**

**#include <mpi/mpi.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#define ROWS\_A 2048**

**#define ROWS\_B 2048**

**#define COLS\_A ROWS\_B**

**#define COLS\_B 2048**

**#define X 1**

**#define Y 0**

**#define DIMENSION 2**

**void part\_multiply(double \*c, const double \*a, const double \*b, int sizeRow, int sizeColumn) {**

**for (int i = 0; i < sizeRow; ++i) {**

**for (int j = 0; j < sizeColumn; ++j) {**

**c[i \* sizeColumn + j] = 0;**

**for (int k = 0; k < ROWS\_B; ++k) {**

**c[i \* sizeColumn + j] += a[i \* ROWS\_B + k] \* b[k \* sizeColumn + j];**

**}**

**}**

**}**

**}**

**double \* generate\_double\_arr(size\_t n){**

**return (double \*) malloc(n \* sizeof(double));**

**}**

**double generate\_random\_in\_range(double min, double max) {**

**double range = (max - min);**

**double div = RAND\_MAX / range;**

**return min + (rand() / div);**

**}**

**double fill\_matrix\_random\_values(double \*matrix, size\_t rows, size\_t cols){**

**for(size\_t i = 0; i < rows; i++){**

**for(size\_t j = 0; j < cols; j++){**

**//matrix[i\*cols + j] = generate\_random\_in\_range(10, 100);**

**matrix[i\*cols + j] = (double)((i) \* cols + j);**

**}**

**}**

**}**

**void print\_matrix(double \*arr, size\_t rows, size\_t cols){**

**for(size\_t i = 0; i < rows; i++){**

**for(size\_t j = 0; j < cols; j++){**

**printf("%lf ", arr[i\*cols + j]);**

**//printf("%zu\n", i\*rows + j);**

**}**

**printf("\n");**

**}**

**printf("\n\n");**

**}**

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

**srand (time(NULL));**

**int rank, size;**

**double \*a, \*b, \*c;**

**MPI\_Init(NULL, NULL);**

**double start\_time = MPI\_Wtime();**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**int dims[DIMENSION]={2,12}, periods[DIMENSION]={0,0};**

**// fill array dims dims[0] \* dims[1] = size**

**MPI\_Dims\_create(size,DIMENSION, dims);**

**MPI\_Comm custom\_communicator;**

**MPI\_Cart\_create(MPI\_COMM\_WORLD, DIMENSION, dims, periods, 0, &custom\_communicator);**

**int custom\_rank;**

**MPI\_Comm\_rank(custom\_communicator, &custom\_rank);**

**int coords[DIMENSION];**

**MPI\_Cart\_coords(custom\_communicator, rank, DIMENSION, coords);**

**//printf("Coords %d %d\n", coords[0], coords[1]);**

**// A 2 строки 4 столбца**

**if(coords[Y] == 0 && coords[X] == 0){**

**//printf("Num procs: %d\n", size);**

**a = generate\_double\_arr(ROWS\_A\*COLS\_A);**

**fill\_matrix\_random\_values(a, ROWS\_A, COLS\_A);**

**b = generate\_double\_arr(ROWS\_B\*COLS\_B);**

**fill\_matrix\_random\_values(b, ROWS\_B, COLS\_B);**

**c = generate\_double\_arr(ROWS\_A\*COLS\_B);**

**// printf("Matrix a\n");**

**// print\_matrix(a, ROWS\_A, COLS\_A);**

**// printf("Matrix b\n");**

**// print\_matrix(b, ROWS\_B, COLS\_B);**

**}**

**int numberRows = ROWS\_A / dims[Y];**

**int numberColumns = COLS\_B / dims[X];**

**double \*partA = generate\_double\_arr(numberRows \* COLS\_A);**

**double \*partB = generate\_double\_arr(numberColumns \* ROWS\_B);**

**double \*partC = generate\_double\_arr(numberColumns \* numberRows);**

**int rowsFlags[2]={0, 1};**

**MPI\_Comm rows;**

**MPI\_Cart\_sub(custom\_communicator, rowsFlags, &rows);**

**int colsFlags[2]={1, 0};**

**MPI\_Comm cols;**

**MPI\_Cart\_sub(custom\_communicator, colsFlags, &cols);**

**if(coords[X] == 0){**

**MPI\_Scatter(a, numberRows \* COLS\_A, MPI\_DOUBLE, partA, numberRows \* COLS\_A, MPI\_DOUBLE, 0, cols);**

**// if(rank == 0){**

**// printf("Number of rows %d, %d %d\n", numberRows, dims[0], dims[1]);**

**// print\_matrix(partA, numberRows, COLS\_A);**

**// }**

**}**

**MPI\_Bcast(partA, numberRows \* COLS\_A, MPI\_DOUBLE, 0, rows);**

**// if(rank == 3){**

**// print\_matrix(partA, numberRows, COLS\_A);**

**// }**

**MPI\_Datatype column;**

**MPI\_Datatype resized\_column;**

**MPI\_Type\_vector(ROWS\_B, numberColumns, COLS\_B, MPI\_DOUBLE, &column);**

**MPI\_Type\_commit(&column);**

**MPI\_Type\_create\_resized(column, 0, (size\_t)(numberColumns \* sizeof(double)), &resized\_column);**

**MPI\_Type\_commit(&resized\_column);**

**if(coords[Y] == 0){**

**MPI\_Scatter(b, 1, resized\_column, partB, numberColumns \* ROWS\_B, MPI\_DOUBLE, 0, rows);**

**}**

**// if(rank == 1){**

**// print\_matrix(partB, ROWS\_B, numberColumns);**

**// }**

**MPI\_Bcast(partB, numberColumns \* ROWS\_B, MPI\_DOUBLE, 0, cols);**

**// if(rank == 3){**

**// print\_matrix(partB, ROWS\_B, numberColumns);**

**// }**

**part\_multiply(partC, partA, partB, numberRows, numberColumns);**

**// if(rank == 0){**

**// printf("Matrix part c\n");**

**// print\_matrix(partC, numberRows, numberColumns);**

**// printf("====\n");**

**// }**

**MPI\_Datatype block;**

**MPI\_Datatype block\_resized;**

**MPI\_Type\_vector(numberRows, numberColumns, COLS\_B, MPI\_DOUBLE, &block);**

**MPI\_Type\_commit(&block);**

**MPI\_Type\_create\_resized(block, 0, (size\_t)(numberColumns \* sizeof(double)), &block\_resized);**

**MPI\_Type\_commit(&block\_resized);**

**int recv\_counts[size];**

**for(size\_t i = 0; i < size; i++){**

**recv\_counts[i] = 1;**

**}**

**int displs[size];**

**for (int i = 0; i < dims[Y]; ++i) {**

**for (int j = 0; j < dims[X]; ++j) {**

**displs[i \* dims[X] + j] = i \* dims[X] \* numberRows + j;**

**}**

**}**

**MPI\_Gatherv(partC, numberColumns\*numberRows, MPI\_DOUBLE, c, recv\_counts, displs, block\_resized, 0, custom\_communicator);**

**if(custom\_rank == 0){**

**//print\_matrix(c, ROWS\_A, COLS\_B);**

**free(a);**

**free(b);**

**free(c);**

**}**

**free(partA);**

**free(partB);**

**free(partC);**

**MPI\_Type\_free(&column);**

**MPI\_Type\_free(&resized\_column);**

**MPI\_Type\_free(&block);**

**MPI\_Type\_free(&block\_resized);**

**double end\_time = MPI\_Wtime();**

**if(rank == 0){**

**printf("%f\n", end\_time-start\_time);**

**}**

**MPI\_Finalize();**

**}**