###### 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 №3

«Умножение матрицы на матрицу в MPI 2D решетка»

студентки 2 курса, … группы

**…**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

…

Новосибирск 2021

**Содержание**

[Задание 3](#_Toc73019545)

[Описание алгоритма 4](#_Toc73019546)

[Таблицы зависимости времени работы программ от размера матриц и размера решётки 5](#_Toc73019547)

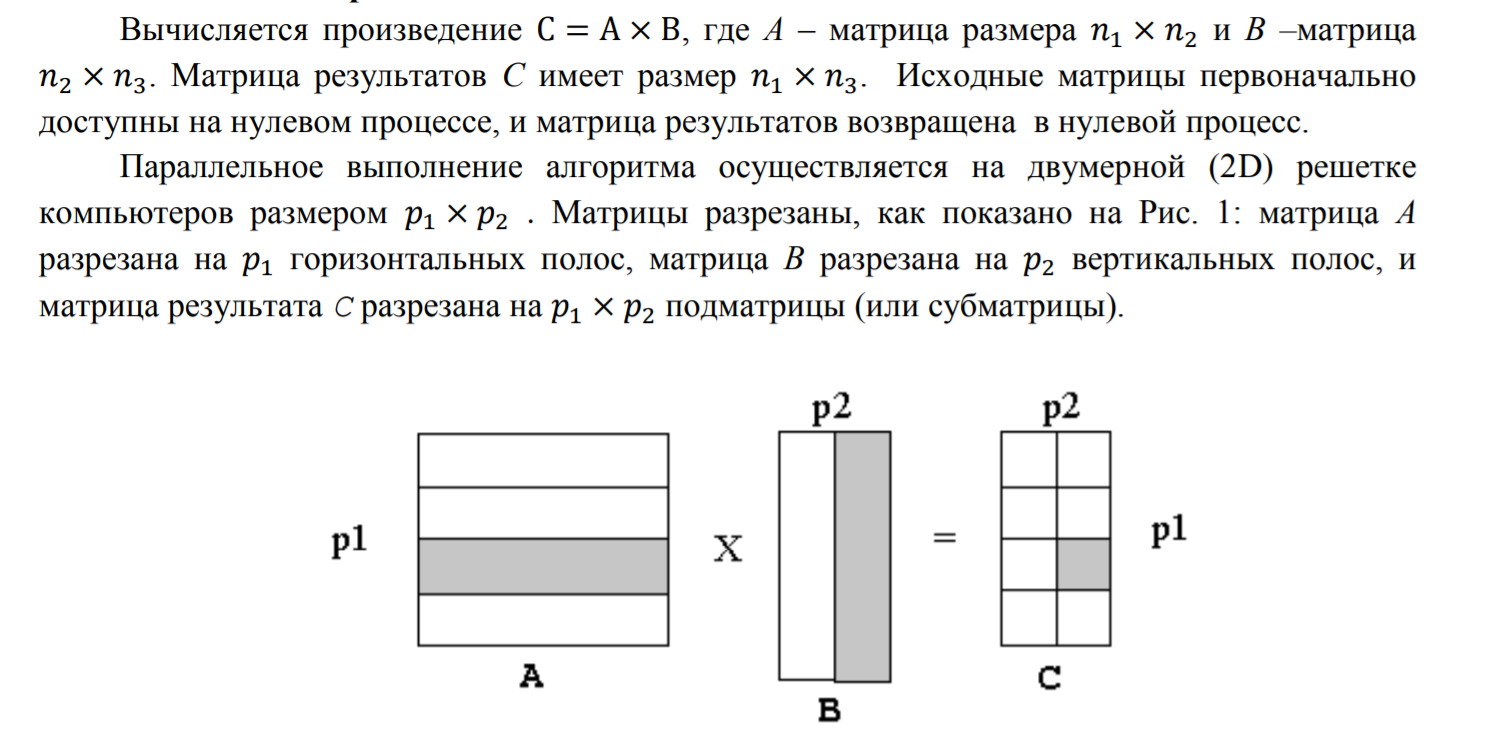
[Профилирование 6](#_Toc73019548)

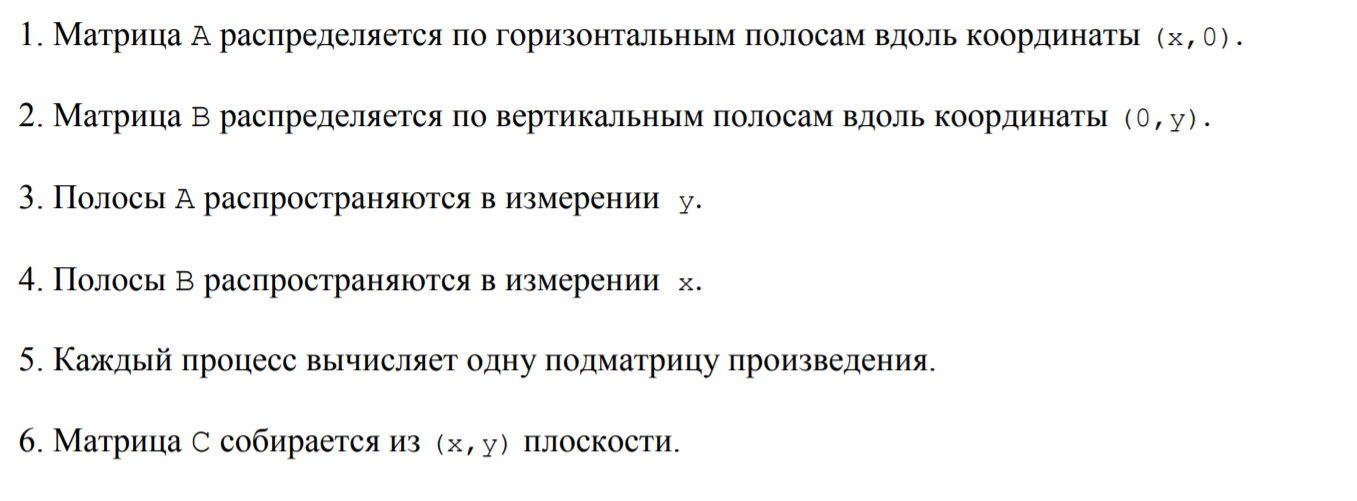
[Листинг программы 7](#_Toc73019549)

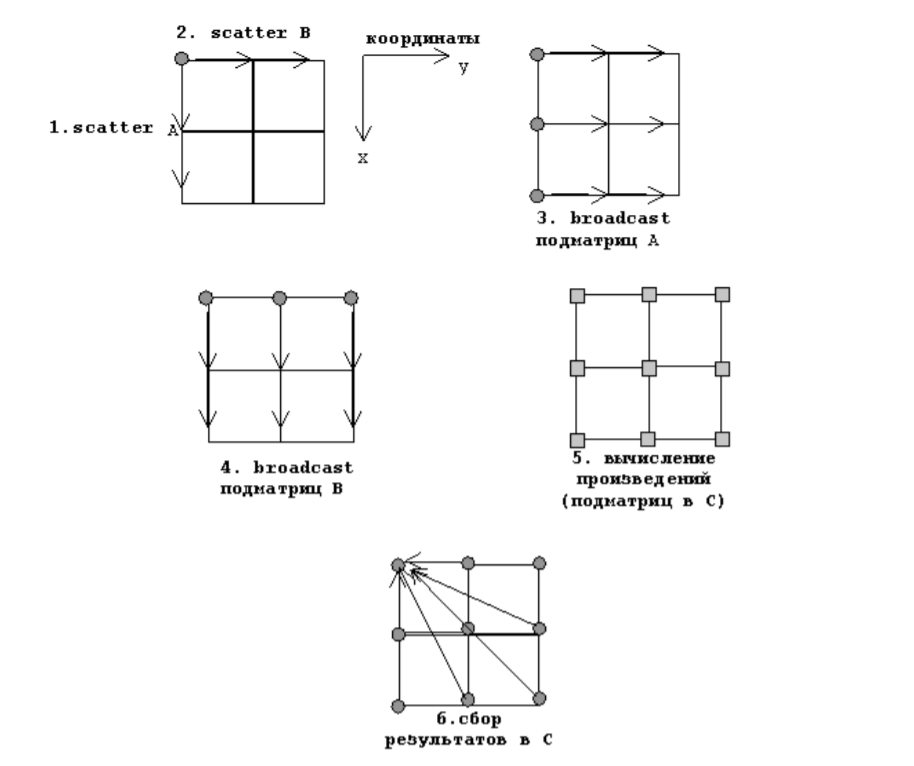
# Задание

* Реализовать параллельный алгоритм умножения матрицы на матрицу при 2D решетке.
* Исследовать производительность параллельной программы в зависимости от размера матрицы и размера решетки.
* Выполнить профилирование программы с помощью MPE при использовании 16-и ядер.

# Описание алгоритма







# Таблицы зависимости времени работы программ от размера матриц и размера решётки

1600 \* 1600 \* 1600

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sizex  Sizey | 1 | 2 | 4 | 8 | 16 |
| 1 | 50.7 | 21.3 | 10.9 | 4.9 |  |
| 2 | 25.5 | 13 | 6.8 |  |  |
| 4 | 13.2 | 6.9 |  |  |  |
| 8 | 6.8 |  |  |  |  |
| 16 |  |  |  |  |  |

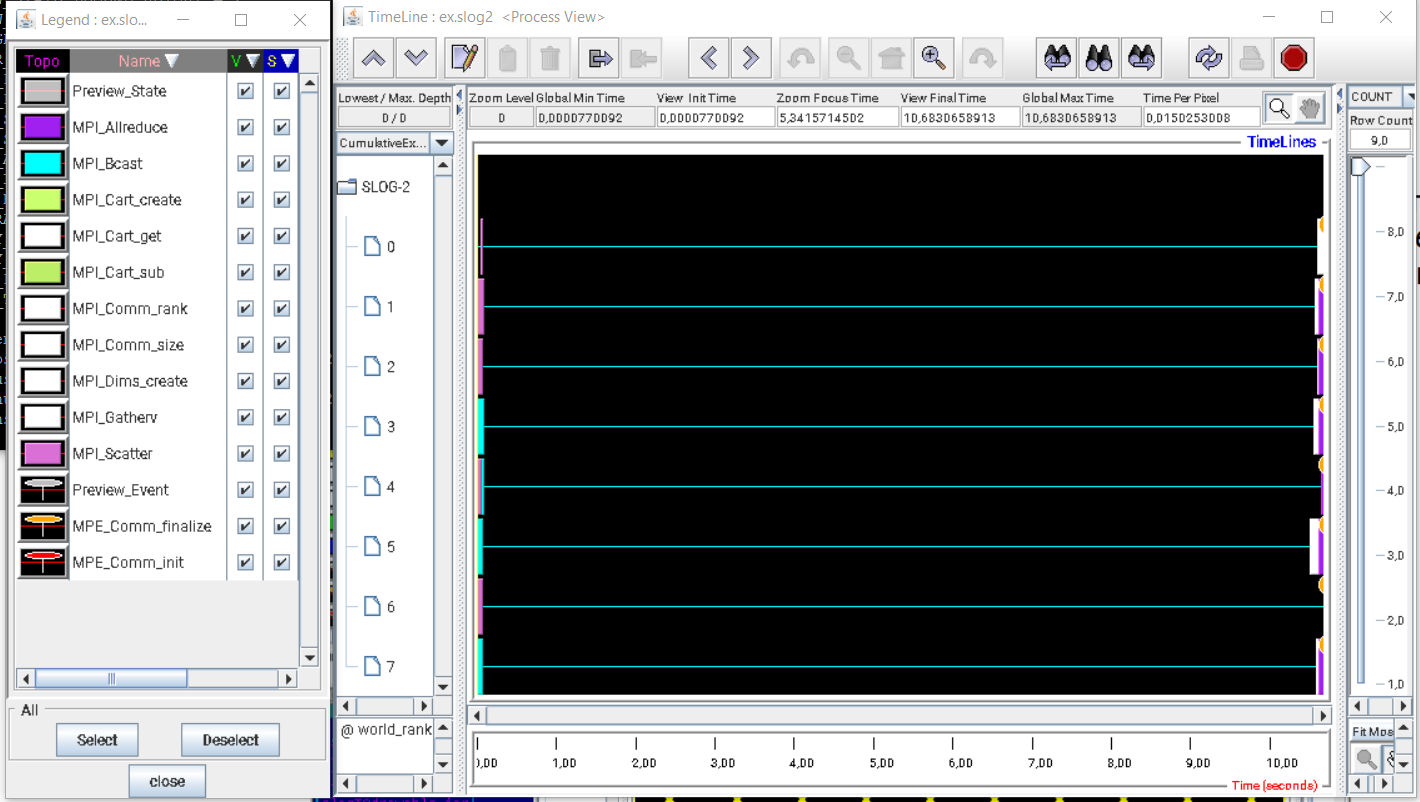
400 \* 15000 \* 600

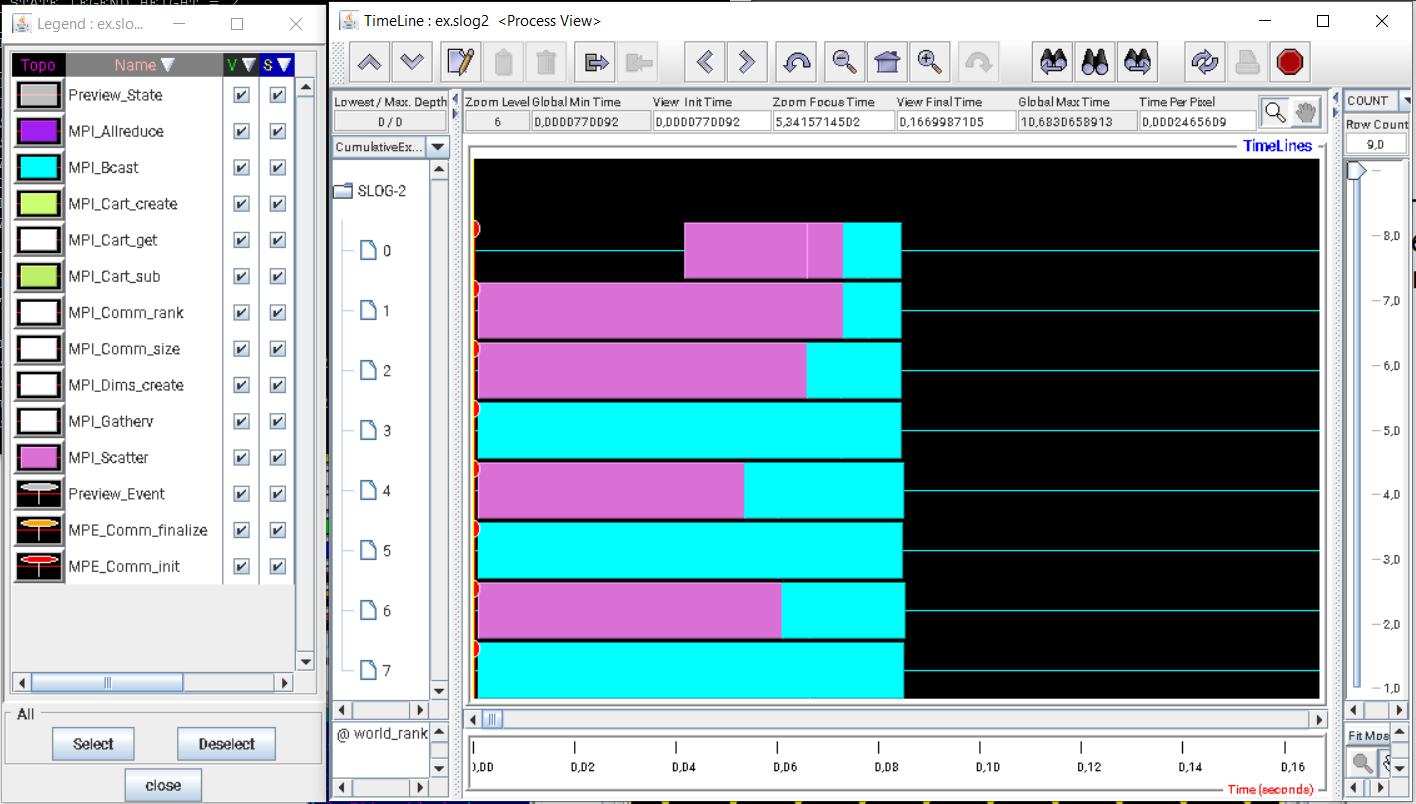
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sizex  Sizey | 1 | 2 | 4 | 8 | 16 |
| 1 | 45.2 | 24.1 | 14.9 | 7.3 |  |
| 2 | 22.7 | 12.4 | 7.7 |  |  |
| 4 | 11.7 | 6.7 |  |  |  |
| 8 | 6.3 |  |  |  |  |
| 16 |  |  |  |  |  |

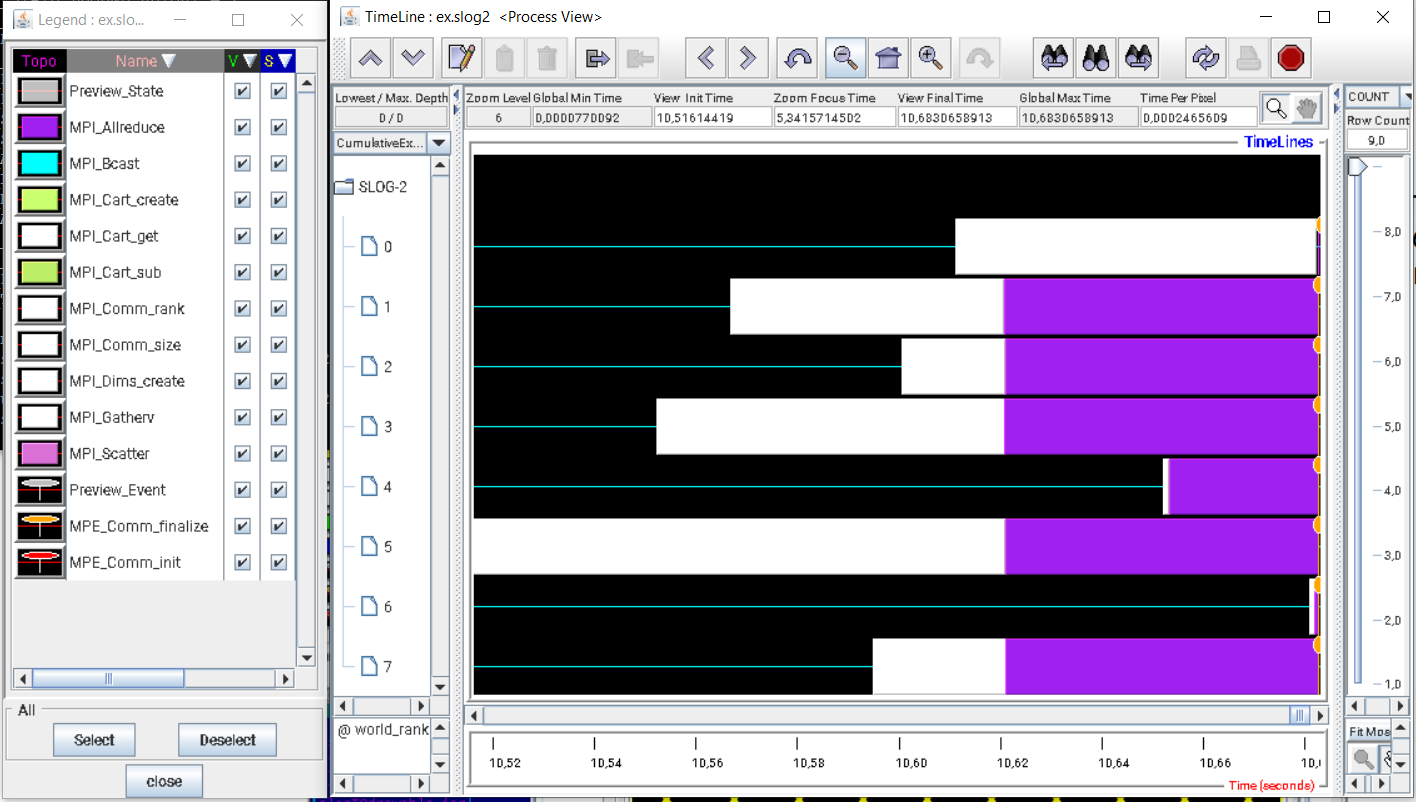
4000 \* 2000 \* 800

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sizex  Sizey | 1 | 2 | 4 | 8 | 16 |
| 1 | 72 | 32.8 | 13.7 | 7.3 |  |
| 2 | 36.7 | 18.6 | 8.4 |  |  |
| 4 | 20.4 | 10.7 |  |  |  |
| 8 | 10.6 |  |  |  |  |
| 16 |  |  |  |  |  |

# Профилирование







# Листинг программы

#include <mpi.h>

#include <iostream>

#include <math.h>

#include <cstdlib>

#include <ctime>

using namespace std;

double\* mul\_matrix(double \*A, double \*B, int N1, int N2, int N3){

double\* C = new double[N1 \* N3];

for (int i = 0; i < N1 \* N3; i++)

C[i] = 0;

for (int i = 0; i < N1; i++)

for (int k = 0; k < N3; k++)

for (int j = 0; j < N2; j++)

C[i\*N3 + k] += A[i\*N2 + j] \* B[j\*N3 + k];

return C;

}

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

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &start);

int size, rank;

MPI\_Init(&argc, &argv);

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

int dims[2] = {0, 0}, sizey, sizex; //sizey и sizex

// определение размеров решетки: dims

MPI\_Dims\_create(size, 2, dims);

sizey = dims[0];

sizex = dims[1];

int periods[2] = {0,0}; //массив размера ndims для задания граничных условий

//(true - периодические, false - непериодические);

int coords[2];

int reorder = 1; //производить перенумерацию процессов (true) или нет (false);

int ranky, rankx;

// создание коммуникатора: comm2d

MPI\_Comm comm2d; // коммуникатор

MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims, periods, reorder, &comm2d);

// получение своего номера в comm2d: rank

MPI\_Comm\_rank(comm2d, &rank);

// получение своих координат в двумерной решетке: coords

MPI\_Cart\_get(comm2d, 2, dims, periods, coords);

ranky = coords[0];

rankx = coords[1];

MPI\_Comm col\_comm, row\_comm;

int remain\_dims[2];

//коммутаторы для строк

remain\_dims[0] = 0;

remain\_dims[1] = 1;

MPI\_Cart\_sub(comm2d, remain\_dims, &row\_comm);

//коммутаторы для столбцов

remain\_dims[0] = 1;

remain\_dims[1] = 0;

MPI\_Cart\_sub(comm2d, remain\_dims, &col\_comm);

int N1 = 400, N2 = 15000, N3 = 600;

double\* A;

double\* B;

if (rank == 0) {

A = new double[N1 \* N2];

B = new double[N2 \* N3];

for (int i = 0; i < N1 \* N2; ++i)

A[i] = i;

for (int i = 0; i < N2 \* N3; ++i)

B[i] = 2\*i;

}

int M1 = N1 / sizey;

int M3 = N3 / sizex;

double\* part\_A = new double[M1 \* N2];

double\* part\_B = new double[N2 \* M3];

MPI\_Datatype col\_vec, col\_vec\_B;

MPI\_Type\_vector(N2, M3, N3, MPI\_DOUBLE, &col\_vec);

//регистрируем новый тип данных

MPI\_Type\_commit(&col\_vec);

//устанавливаем размер type\_vector'а

MPI\_Type\_create\_resized(col\_vec, 0, M3 \* sizeof(double), &col\_vec\_B);

MPI\_Type\_commit(&col\_vec\_B);

if (rank % sizex == 0)

MPI\_Scatter(A, M1 \* N2, MPI\_DOUBLE, part\_A, M1 \* N2, MPI\_DOUBLE, 0, col\_comm);

if (rank < sizex)

MPI\_Scatter(B, 1, col\_vec\_B, part\_B, M3 \* N2, MPI\_DOUBLE, 0, row\_comm);

MPI\_Bcast(part\_A, M1 \* N2, MPI\_DOUBLE, 0, row\_comm);

MPI\_Bcast(part\_B, M3 \* N2, MPI\_DOUBLE, 0, col\_comm);

double\* part\_C = mul\_matrix(part\_A, part\_B, M1, N2, M3);

delete[] part\_A;

delete[] part\_B;

MPI\_Type\_free(&col\_vec);

MPI\_Type\_free(&col\_vec\_B);

double\* C;

if (rank == 0)

C = new double[N1 \* N3];

MPI\_Datatype m\_vector;

MPI\_Type\_vector(M1, M3, N3, MPI\_DOUBLE, &m\_vector);

MPI\_Type\_commit(&m\_vector);

MPI\_Datatype m\_vector\_C;

MPI\_Type\_create\_resized(m\_vector, 0, M3 \* sizeof(double), &m\_vector\_C);

MPI\_Type\_commit(&m\_vector\_C);

int\* counts = new int[size];

int\* displs = new int[size];

for (int i = 0; i < sizey; i++)

for (int j = 0; j < sizex; j++) {

displs[i \* sizex + j] = j + i \* sizex \* M1;

counts[i \* sizex + j] = 1;

}

MPI\_Gatherv(part\_C, M1 \* M3, MPI\_DOUBLE, C, counts , displs , m\_vector\_C, 0, comm2d);

delete[] counts;

delete[] displs;

delete[] part\_C;

MPI\_Type\_free(&m\_vector);

MPI\_Type\_free(&m\_vector\_C);

if (rank == 0) {

delete[] A;

delete[] B;

delete[] C;

}

clock\_gettime(CLOCK\_MONOTONIC\_RAW, &end);

double time = end.tv\_sec-start.tv\_sec + 0.000000001\*(end.tv\_nsec-start.tv\_nsec);

double fulltime = 0;

MPI\_Allreduce(&time, &fulltime, 1, MPI\_DOUBLE, MPI\_MAX, MPI\_COMM\_WORLD);

if (rank == 0) {

cout << "size = " << size << ", size\_x = " << sizex << ", size\_y = " << sizey << endl;

cout << " N1 = " << N1 << ", N2 = " << N2 << ", N3 = " << N3 << endl;

cout << " Time taken: " << fulltime << " sec." << endl << endl;

}

MPI\_Finalize();

return 0;

}