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 ПРАКТИЧЕСКОЙ РАБОТЫ

«Умножение матрицы на матрицу в MPI 2D решетке»

студентки 2 курса, группы 22204

Клочихиной Софьи Павловны

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А. Ю. Власенко

Новосибирск 2024

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Освоить концепции MPI-коммуникаторов и декартовых топологий, а также концепции производных типов данных.

ЗАДАНИЕ

1. Реализовать параллельный алгоритм умножения матрицы на матрицу при 2D решетке процессов с соблюдением требований.

2. 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.

3. Выполнить профилирование программы при использовании 8-и ядер с решетками 2x4, 4x2.

Общий алгоритм:

1. Создание решетки процессов p1 x p2.

2. Генерация матриц A[n1 x n2] и B[n2 x n3] на процессе с координатами (0;0) как одномерных массивов.

3. Раздача матрицы A по горизонтальным полосам на вертикальную линейку процессов (0;0), (1;0), (2;0), ..., (p1 - 1; 0) при помощи MPI_Scatter.

4. Определение нового производного типа данных для выбора из матрицы B вертикальных полос.

5. Раздача матрицы B по вертикальным полосам на горизонтальную линейку процессов (0;0), (0;1), (0;2), ..., (0; p2 – 1) таким образом, что каждому процессу высылается только 1 элемент производного типа.

6. Каждый из процессов в левой вертикальной колонке ((1;0), (2;0), ..., (p1 - 1; 0)) при помощи MPI_Bcast раздает свою полосу матрицы A всем процессам своей горизонтали. Т.е. процесс (1;0) раздает свою полосу процессам (1;1), (1;2),...

7. То же с полосами матрицы B, которые процессы первой горизонтали раздают по своим вертикальным столбцам решетки процессов (MPI_Bcast).

8. Теперь на каждом процессе есть по полосе A и по столбцу B, перемножаем, получаем миноры C.

9. Собираем всю C на процессе (0;0).

ОПИСАНИЕ РАБОТЫ

1. Была написана программа с использованием технологии MPI. Время выполнения было замерено с помощью функции MPI_Wtime();

2. Был написан скрипт с использованием SLURM, с помощью которого программа компилировалась и запускалась:

```
[fit_opp@gpn-headnode lab4]$ sbatch ./cart.sbatch 6 4
Submitted batch job 72894
```

3. Время работы программы с разными входными данными получилось следующим:

```
CART: 2 x 12
Time: 28.9114
```

```
CART: 3 x 8
Time: 30.9292
```

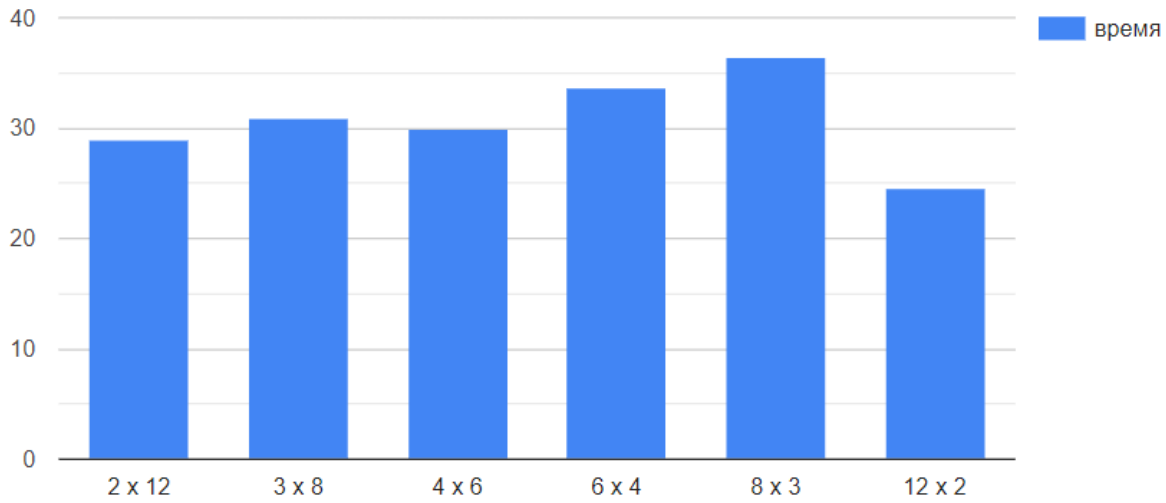
```
CART: 4 x 6
Time: 29.9925
```

```
CART: 6 x 4
Time: 33.6727
```

```
CART: 8 x 3
Time: 36.4671
```

```
CART: 12 x 2
Time: 24.6152
```

4. Сравнение времени работы программы на разных входных данных:



5. Было выполнено профилирование на 8-ми процессах:
   Для решётки 2 x 4:

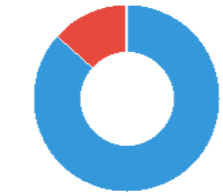
Summary: cart.2.4.stf

Total time: **73.7** sec. Resources: **8** processes, **1** node.

Continue >

Ratio

This section represents a ratio of all MPI calls to the rest of your code in the application.



Serial Code - 63.8 sec 86.5 %

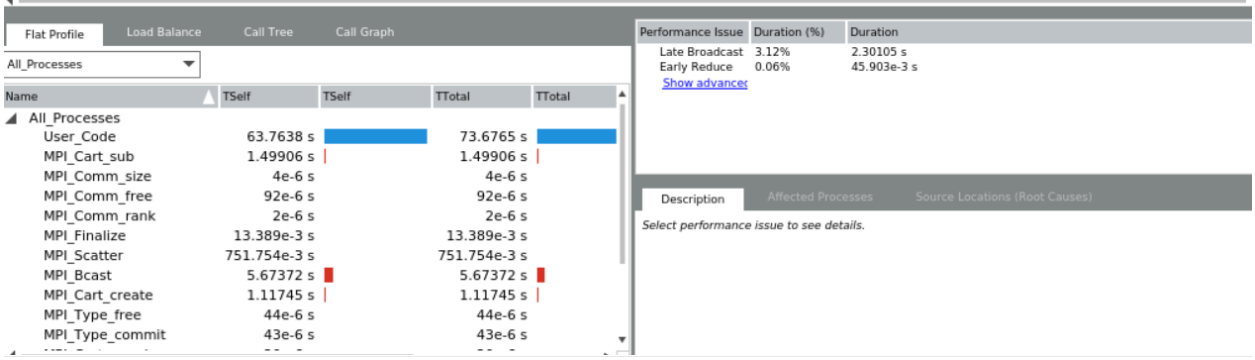
OpenMP - 0 sec 0 %

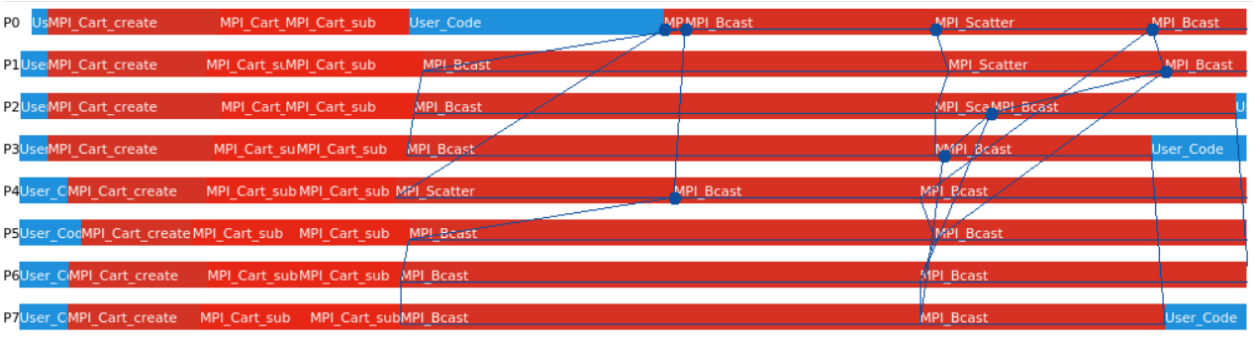
MPI calls - 9.91 sec 13.4 %

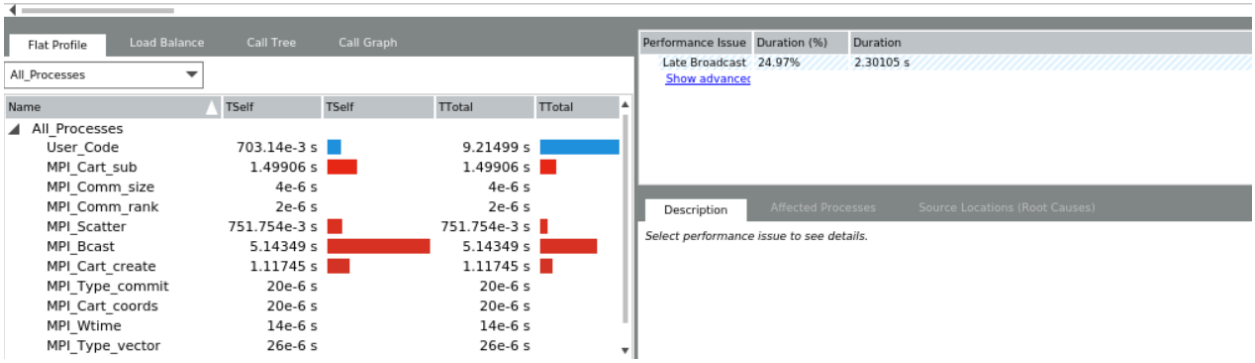
Exclude from total time

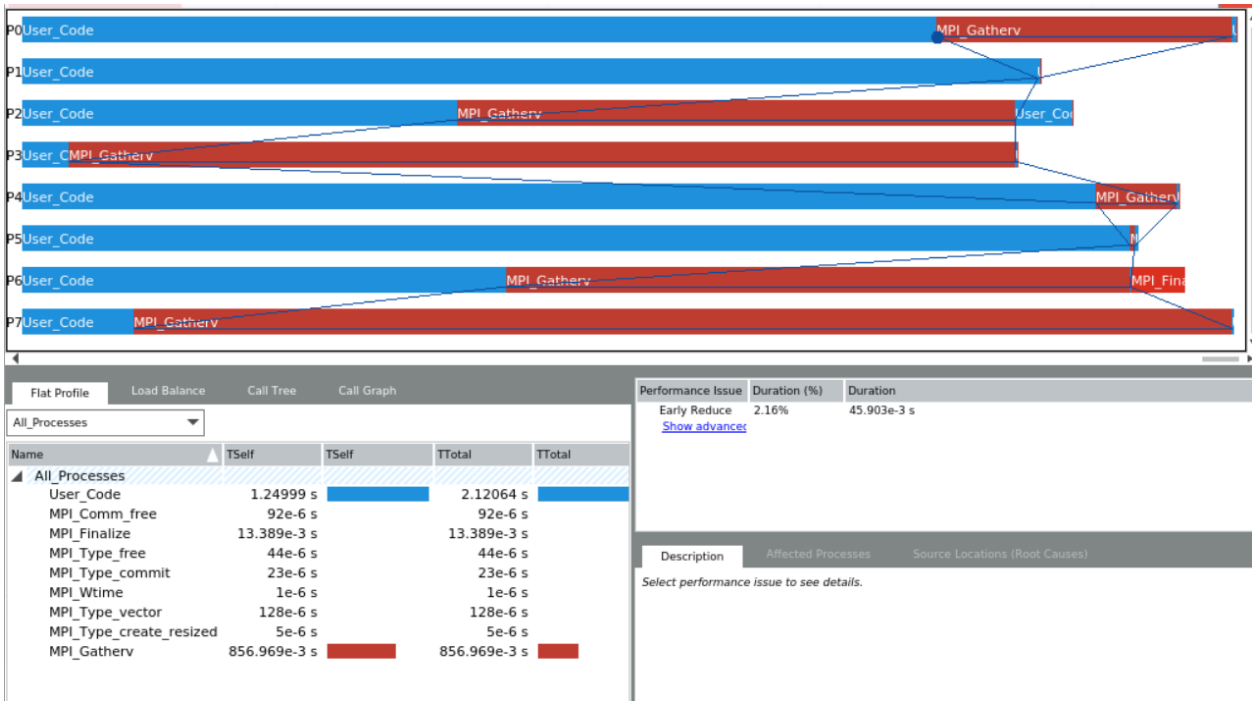
Top MPI functions

This section lists the most active MPI functions from all MPI calls in the application.

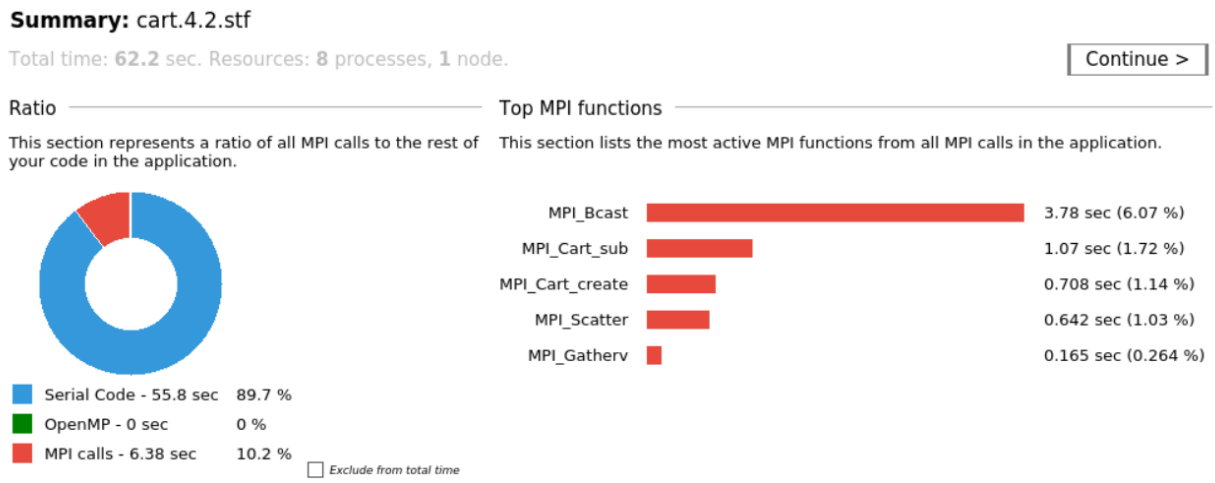


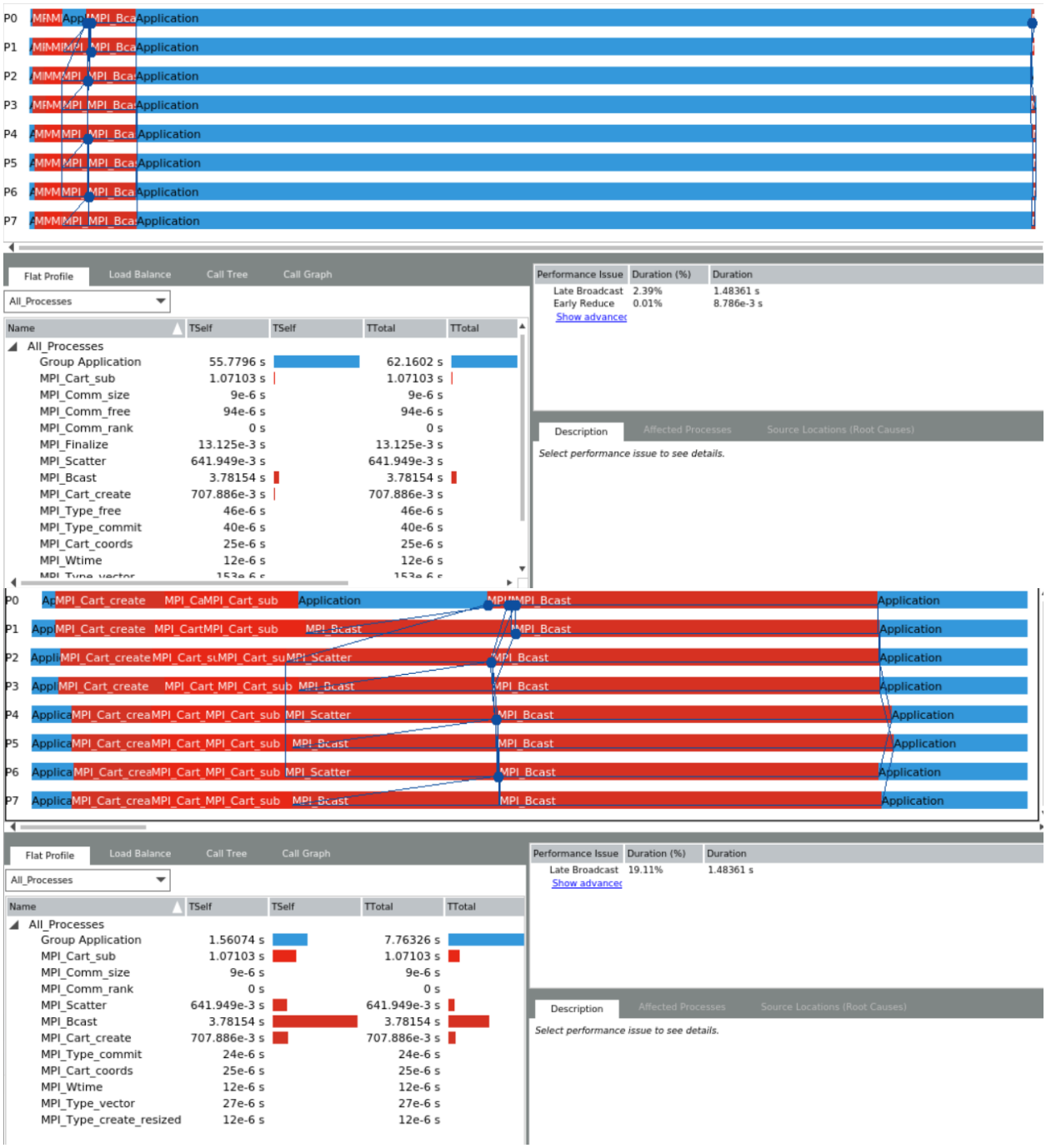






Для решётки 4 x 2:




P0
P1
P2
P3
P4
P5
P6
P7
Application
Flat Profile
Load Balance
Call Tree
Call Graph
All_Processes
Name
TSelf
TTotal
All_Processes
Group Application 55.7796 s 62.1602 s
MPI_Cart_sub 1.07103 s 1.07103 s
MPI_Comm_size 9e-6 s 9e-6 s
MPI_Comm_free 94e-6 s 94e-6 s
MPI_Comm_rank 0 s 0 s
MPI_Finalize 13.125e-3 s 13.125e-3 s
MPI_Scatter 641.949e-3 s 641.949e-3 s
MPI_Bcast 3.78154 s 3.78154 s
MPI_Cart_create 707.886e-3 s 707.886e-3 s
MPI_Type_free 46e-6 s 46e-6 s
MPI_Type_commit 40e-6 s 40e-6 s
MPI_Cart_coords 25e-6 s 25e-6 s
MPI_Wtime 12e-6 s 12e-6 s
Performance Issue
Duration (%)
Duration
Late Broadcast 2.39% 1.48361 s
Early Reduce 0.01% 8.786e-3 s
Show advanced
Description
Affected Processes
Source Locations (Root Causes)
Select performance issue to see details.
MPI_Cart_create
MPI_Cart_sub
MPI_Scatter
MPI_Bcast
Flat Profile
Load Balance
Call Tree
Call Graph
All_Processes
Name
TSelf
TTotal
All_Processes
Group Application 1.56074 s 7.76326 s
MPI_Cart_sub 1.07103 s 1.07103 s
MPI_Comm_size 9e-6 s 9e-6 s
MPI_Comm_rank 0 s 0 s
MPI_Scatter 641.949e-3 s 641.949e-3 s
MPI_Bcast 3.78154 s 3.78154 s
MPI_Cart_create 707.886e-3 s 707.886e-3 s
MPI_Type_commit 24e-6 s 24e-6 s
MPI_Cart_coords 25e-6 s 25e-6 s
MPI_Wtime 12e-6 s 12e-6 s
MPI_Type_vector 27e-6 s 27e-6 s
MPI_Type_create_resized 12e-6 s 12e-6 s
Performance Issue
Duration (%)
Duration
Late Broadcast 19.11% 1.48361 s
Show advanced
Description
Affected Processes
Source Locations (Root Causes)
Select performance issue to see details.


P0 Application MPI_Gatherv
P1 Application MPI_Gatherv MPI_Finalize
P2 Application MPI_Gatherv Application
P3 Application MPI_Gatherv
P4 Application MPI_Gatherv
P5 Application MPI_Gatherv
P6 Application MPI_Gatherv
P7 Application MPI_Gatherv
Flat Profile
Load Balance
Call Tree
Call Graph
All_Processes
Name
TSelf
TSelf
TTotal
TTotal
All_Processes
Group Application 227.294e-3 s 405.338e-3 s
MPI_Comm_free 94e-6 s 94e-6 s
MPI_Finalize 13.125e-3 s 13.125e-3 s
MPI_Type_free 46e-6 s 46e-6 s
MPI_Type_commit 16e-6 s 16e-6 s
MPI_Wtime 0 s 0 s
MPI_Type_vector 126e-6 s 126e-6 s
MPI_Type_create_resized 5e-6 s 5e-6 s
MPI_Gatherv 164.632e-3 s 164.632e-3 s
Performance Issue
Duration (%)
Duration
Early Reduce 2.17% 8.786e-3 s
Show advanced
Description
Affected Processes
Source Locations (Root Causes)
Select performance issue to see details.

ЗАКЛЮЧЕНИЕ

Были освоены концепции MPI коммуникаторов и декартовых топологий, а также концепции производных типов данных.

Приложение 1. Полный листинг параллельной программы на C++.

```
#include <iostream>
#include <mpi.h>

int N1;
int N2;
int N3;

const int X = 0;
const int Y = 1;

const int NDIMS = 2;

const int n1_mult = 500;
const int n3_mult = 750;

void initNs(const int* dims) {
    N1 = dims[X] * n1_mult;
    N3 = dims[Y] * n3_mult;
    N2 = (N3 + N1) / 2;
}

void generateMatrix(double* matrix, int row, int column) {
    for(int i = 0; i < row; i++)
        for(int j = 0; j < column; j++)
            matrix[i * column + j] = (double)rand() / RAND_MAX * 20.0 - 10.0;
}

void createCartComm(MPI_Comm& cart, int size, int argc, char** argv, int* dims) {
    if (argc <= 2)
        MPI_Dims_create(size, NDIMS, dims);
    else {
        dims[X] = strtol(argv[1], nullptr, 10);
        dims[Y] = strtol(argv[2], nullptr, 10);

        if (dims[X] * dims[Y] != size) exit(EXIT_FAILURE);
    }
    bool reorder = true;
    int periodic[NDIMS] = {};
    MPI_Cart_create(MPI_COMM_WORLD, NDIMS, dims, periodic, reorder, &cart);
}

void createSubComms(MPI_Comm& cart, MPI_Comm& rows, MPI_Comm& columns) {
    int remain_dims[NDIMS];

    remain_dims[X] = false, remain_dims[Y] = true;
    MPI_Cart_sub(cart, remain_dims, &rows);

    remain_dims[X] = true, remain_dims[Y] = false;
    MPI_Cart_sub(cart, remain_dims, &columns);
}

void mult(double* C_part, const double* A_part, const double* B_part, int A_rows,
int B_cols) {
    for (int i = 0; i < A_rows; i++)
        for (int j = 0; j < N2; j++)
            for (int k = 0; k < B_cols; k++)
                C_part[i * B_cols + k] += A_part[i * N2 + j] * B_part[j * B_cols +
k];
}
```

```
void gatherC(double* C, double* C_part, MPI_Comm& cart, int size, int dim_x, int
dim_y) {
    MPI_Datatype C_block, C_blocktype;

    int* recvcounts = new int [size];
    int* displs = new int [size];

    for (int i = 0; i < dim_x; i++)
        for (int j = 0; j < dim_y; j++) {
            recvcounts[i * dim_y + j] = 1;
            displs[i * dim_y + j] = i * n1_mult * dim_y + j;
        }

    MPI_Type_vector(n1_mult, n3_mult, N3, MPI_DOUBLE, &C_block);
    MPI_Type_commit(&C_block);

    MPI_Type_create_resized(C_block, 0, n3_mult * sizeof(double), &C_blocktype);
    MPI_Type_commit(&C_blocktype);

    MPI_Gatherv(C_part, n1_mult * n3_mult, MPI_DOUBLE, C, recvcounts, displs,
C_blocktype, 0, MPI_COMM_WORLD);

    MPI_Type_free(&C_block);
    MPI_Type_free(&C_blocktype);

    delete[] displs;
    delete[] recvcounts;
}

int main(int argc, char** argv) {
    int size, rank;
    double start, end;

    double* A;
    double* B;
    double* C;

    double* A_part;
    double* B_part;
    double* C_part;

    MPI_Comm cart;
    MPI_Comm rows;
    MPI_Comm columns;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int dims[NDIMS] = {};
    int coords[NDIMS] = {};

    createCartComm(cart, size, argc, argv, dims);
    initNs(dims);

    createSubComms(cart, rows, columns);

    MPI_Cart_coords(cart, rank, NDIMS, coords);

    if (!coords[X] && !coords[Y]) {
        A = new double [N1 * N2];
        B = new double [N2 * N3];
```

```
        C = new double [N1 * N3]{};

        generateMatrix(A, N1, N2);
        generateMatrix(B, N2, N3);
    }

    start = MPI_Wtime();

    int A_part_size = n1_mult * N2;
    int B_part_size = N2 * n3_mult;

    A_part = new double [A_part_size];
    B_part = new double [B_part_size];
    C_part = new double [n1_mult * n3_mult];

    if (!coords[Y])
        MPI_Scatter(A, A_part_size, MPI_DOUBLE, A_part, A_part_size, MPI_DOUBLE,
0, columns);
    MPI_Bcast(A_part, A_part_size, MPI_DOUBLE, 0, rows);

    MPI_Datatype B_block, B_blocktype;

    MPI_Type_vector(N2, n3_mult, N3, MPI_DOUBLE, &B_block);
    MPI_Type_commit(&B_block);

    MPI_Type_create_resized(B_block, 0, n3_mult * sizeof(double), &B_blocktype);
    MPI_Type_commit(&B_blocktype);

    if (!coords[X])
        MPI_Scatter(B, 1, B_blocktype, B_part, n3_mult * N2, MPI_DOUBLE, 0, rows);
    MPI_Bcast(B_part, B_part_size, MPI_DOUBLE, 0, columns);

    mult(C_part, A_part, B_part, n1_mult, n3_mult);

    gatherC(C, C_part, cart, size, dims[X], dims[Y]);

    if (!coords[X] && !coords[Y]) {
        end = MPI_Wtime();
        std::cout << "Time: " << end - start << std::endl;
        delete[] A;
        delete[] B;
        delete[] C;
    }

    delete[] A_part;
    delete[] B_part;
    delete[] C_part;

    MPI_Comm_free(&cart);
    MPI_Comm_free(&rows);
    MPI_Comm_free(&columns);

    MPI_Type_free(&B_block);
    MPI_Type_free(&B_blocktype);

    MPI_Finalize();
    return 0;
}
```

Приложение 2. Полный листинг скрипта SLURM для параллельной программы.

```
#!/bin/bash
#SBATCH -J lab4          # Job name
#SBATCH -p compclass     # Queue name (or "compclass_unstable", or "gpuserv", or
"a100serv")
#SBATCH -o lab4.%j.out   # Name of stdout output file (%j expands to %jobId)
#SBATCH -N 1             # Total number of nodes requested
#SBATCH -n 2             # Total number of mpi tasks requested
#SBATCH -t 00:10:00      # Run time (hh:mm:ss) - 1 minute

module load mpi/mpich-x86_64

if [ "$#" -ne 2 ]; then
    echo "Usage: $0 <dimension X size> <dimension Y size>"
    exit 1
fi

dimX=$1
dimY=$2

SIZE=$((dimX * dimY))

mpicxx cart.cpp -o cart

echo -e "CART: $dimX x $dimY\n"

mpirun -np $SIZE ./cart $dimX $dimY
```