###### 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 решетке»

студента 2 курса, 22202 группы

**Бальчинова А.С.**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

В.А. Перепёлкин

Новосибирск 2024

**СОДЕРЖАНИЕ**

[ЦЕЛЬ 3](#_Toc18443921)

[ЗАДАНИЕ 3](#_Toc18443922)

[ОПИСАНИЕ РАБОТЫ 4-10](#_Toc18443923)

[ЗАКЛЮЧЕНИЕ 11](#_Toc18443924)

[Приложение. 12](#_Toc18443925)-16

**ЦЕЛЬ**

Ознакомиться с MPI типами данных, изучить основные функции MPI и проверить эффективность его применения на практике.

# ЗАДАНИЕ

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

# ОПИСАНИЕ РАБОТЫ

**Описание алгоритма:**

Вычисляется произведение С = А × В, где А – матрица размера 𝑛1 × 𝑛2 и В –матрица 𝑛2 × 𝑛3. Матрица результатов С имеет размер 𝑛1 × 𝑛3.

Исходные матрицы первоначально доступны на нулевом процессе, и матрица результатов возвращена в нулевой процесс.

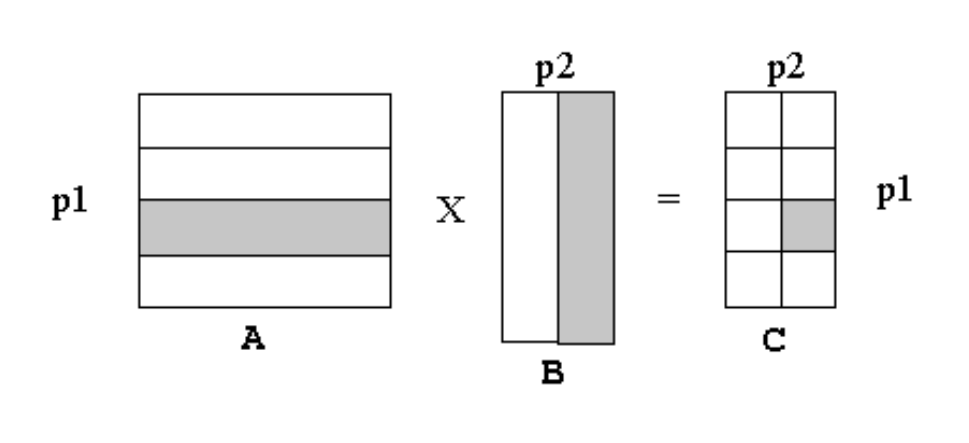
Параллельное выполнение алгоритма осуществляется на двумерной (2D) решетке компьютеров размером 𝑝1 × 𝑝2 .

Матрицы разрезаны, как показано на Рис. 1:

матрица А разрезана на 𝑝1 горизонтальных полос,

матрица В разрезана на 𝑝2 вертикальных полос,

и матрица результата C разрезана на 𝑝1 × 𝑝2 подматрицы (или субматрицы).



**Рис. 1.** Разрезание данных для параллельного алгоритма произведения двух матриц при вычислении в 2D решетке компьютеров. Выделенные данные расположены в одном компьютере

Каждый компьютер (i,j) вычисляет произведение i-й горизонтальной полосы матрицы A и j-й вертикальной полосы матрицы B, произведение получено в подматрице (i,j) матрицы C.

Последовательные стадии вычисления иллюстрируются на Рис. 2:

1. Матрица А распределяется по горизонтальным полосам вдоль координаты (x,0).

2. Матрица B распределяется по вертикальным полосам вдоль координаты (0,y).

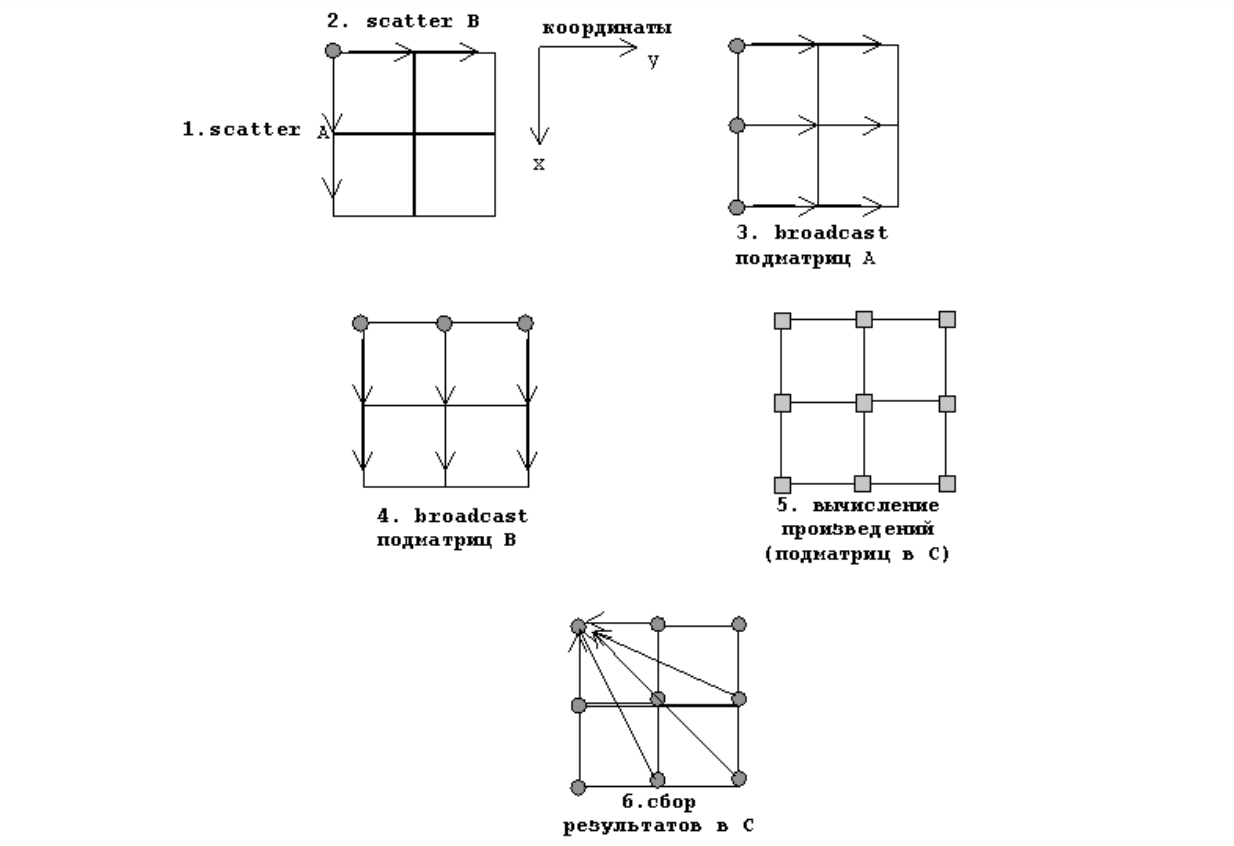
3. Полосы А распространяются в измерении y.

4. Полосы B распространяются в измерении х.

5. Каждый процесс вычисляет одну подматрицу произведения.

6. Матрица C собирается из (x,y) плоскости.

Осуществлять пересылки между компьютерами во время вычислений не нужно, т. к. все полосы матрицы А пересекаются со всеми полосами матрицы B в памяти компьютеров системы.



**Рис. 2** Стадии вычисления произведения матриц в 2D параллельном алгоритме

Затем были произведены замеры времени работы программы в зависимости от размера матрицы и размера решётки.

Размеры матриц:

1) M = N = K = 256

2) M = N = K = 1600

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Решётка, P1\P2 | 1 | 2 | 4 | 8 | 16 |
| 1 | 0,0882071 | 0,0690484 | 0.0329398 | 0.0146679 | 0.0909901 |
| 2 | 0.0803617 | 0.0415428 | 0.0166718 | 0.0822033 |  |
| 4 | 0.0483991 | 0.019022 | 0.0911663 |  |  |
| 8 | 0.0234642 | 0.151196 |  |  |  |
| 16 | 0.194536 |  |  |  |  |

3) M = N = K = 4000

**1**

# 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Решётка, P1\P2 | 1 | 2 | 4 | 8 | 16 |
| 1 | 24.7296 | 25.3089 | 12.2329 | 5.32912 | 2.50268 |
| 2 | 27.145 | 13.4627 | 6.49451 | 3.30491 |  |
| 4 | 13.9642 | 6.75919 | 3.51791 |  |  |
| 8 | 7.08505 | 3.65854 |  |  |  |
| 16 | 4.21351 |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Решётка, P1\P2 | 1 | 2 | 4 | 8 | 16 |
| 1 | 414.888 | 414.925 | 211.946 | 135.479 | 62.6835 |
| 2 | 466.168 | 212.334 | 108.35 | 75.2249 |  |
| 4 | 265.436 | 109.554 | 60.0918 |  |  |
| 8 | 135.605 | 64.8112 |  |  |  |
| 16 | 78.8225 |  |  |  |  |

**3**

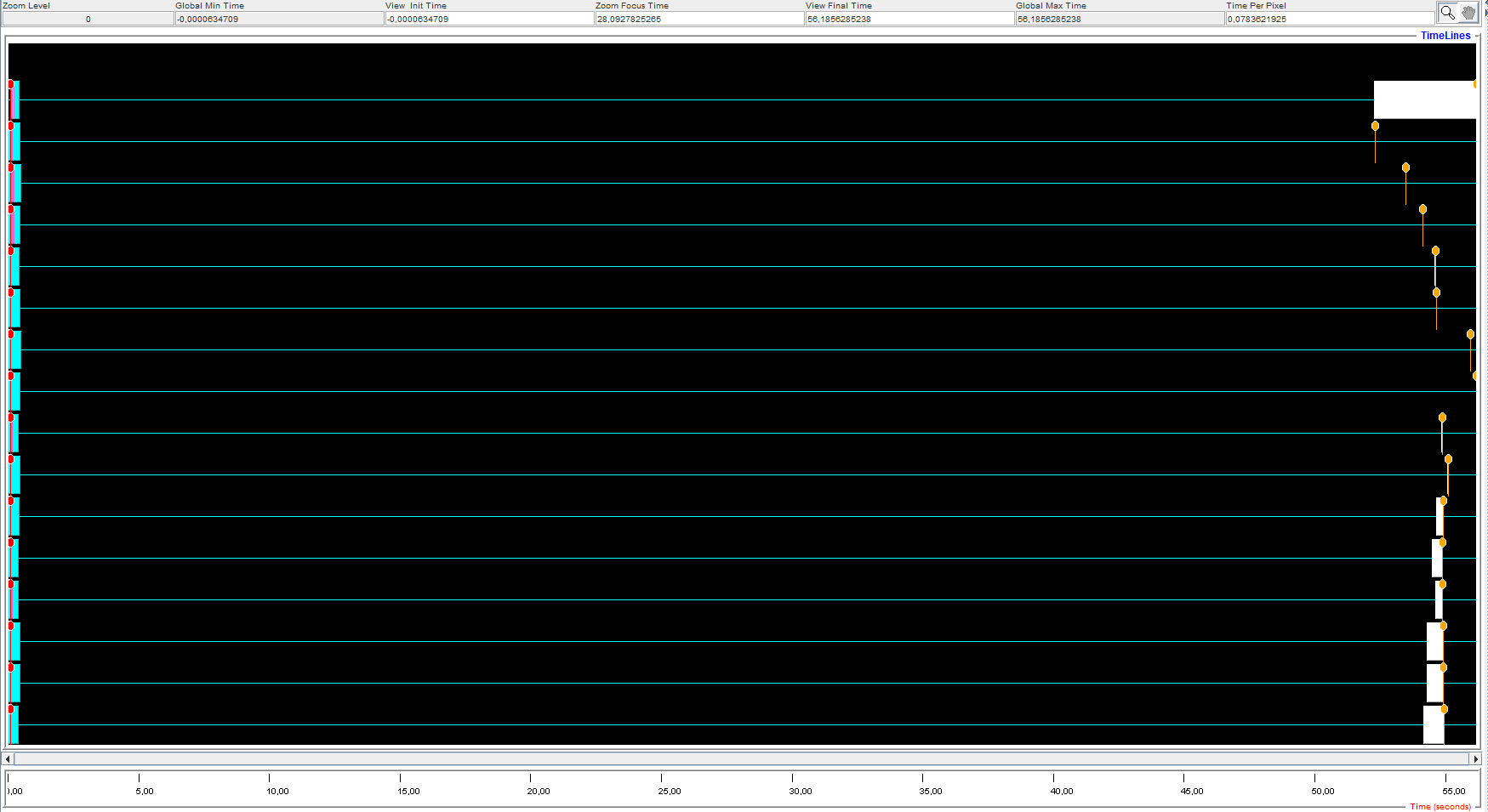
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер | 1x2 | 2x1 | 1x4 | 2x2 | 4x1 |
| Время | 414,925 | 466,168 | 211,946 | 212,334 | 265,436 |

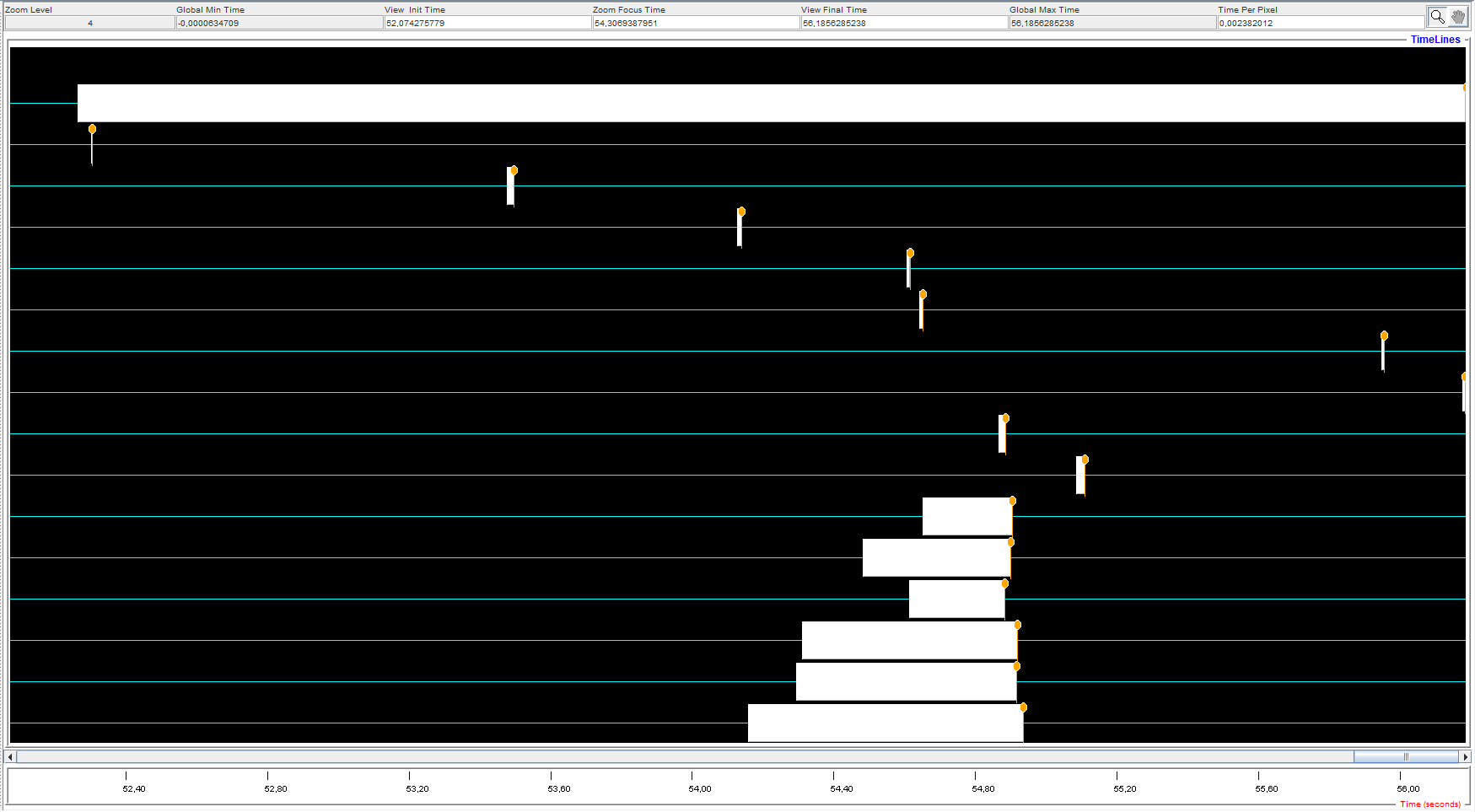
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер | 1x8 | 2x4 | 4x2 | 8x1 |
| Время | 135,479 | 108,35 | 109,554 | 135,605 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер | 1x16 | 2x8 | 4x4 | 8x2 | 16x1 |
| Время | 62,6835 | 75,2249 | 60,0918 | 64,8112 | 78,8225 |

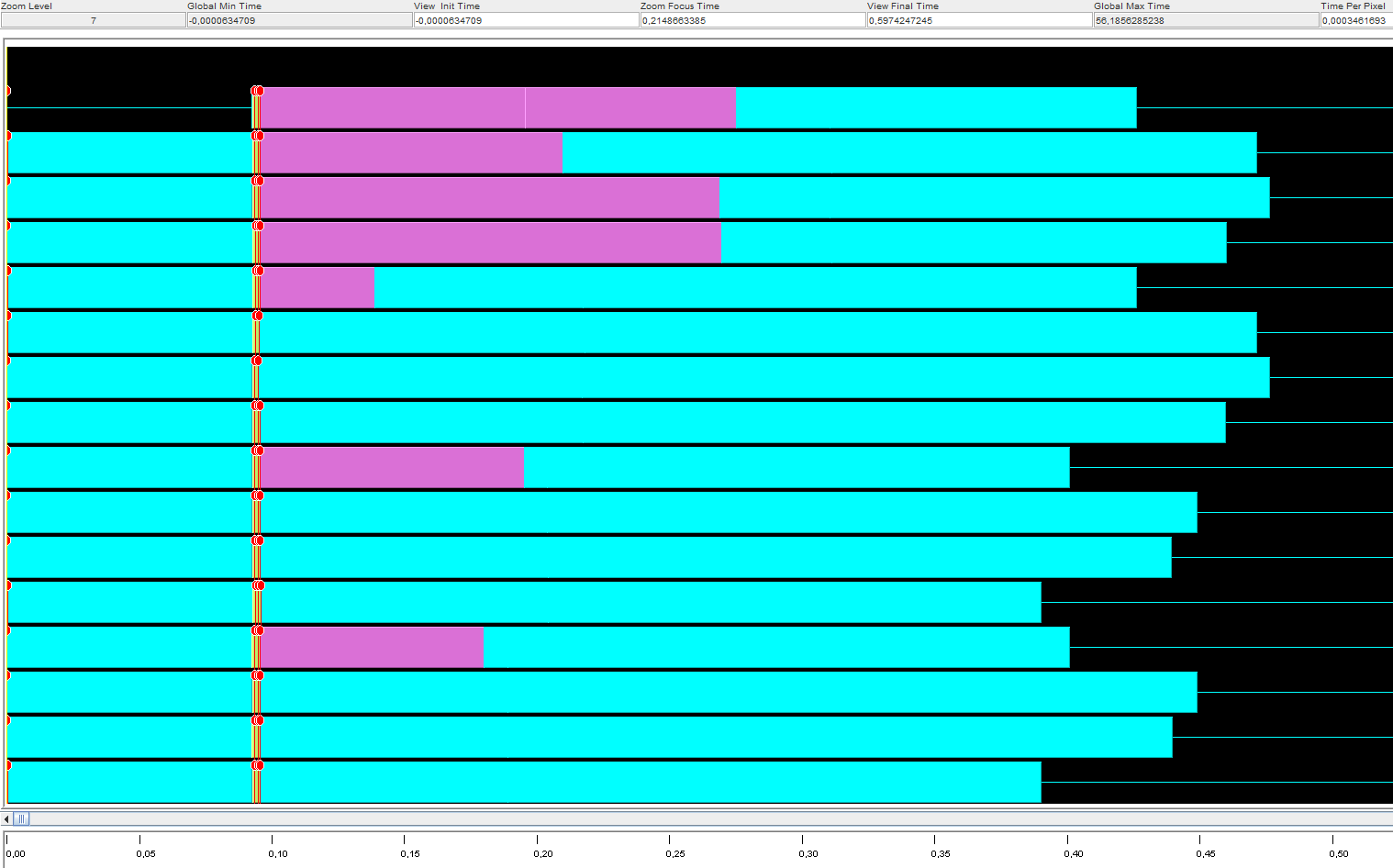
**Профилирование MPE**

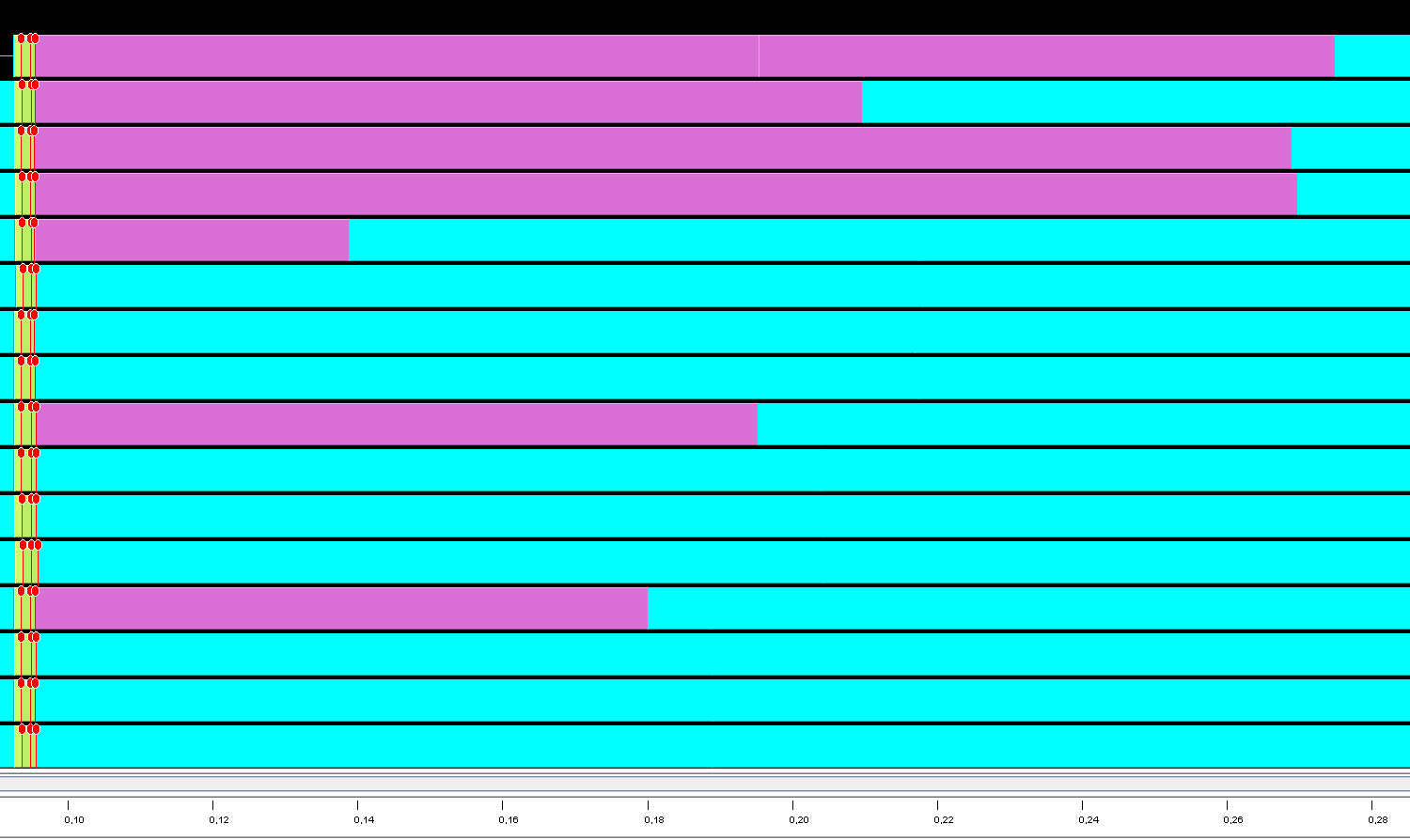












# ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы мы глубже познакомились с MPI, научились применять некоторые его функции на практике, реализовали алгоритм умножения матриц в MPI 2D решётке, а также исследовали производительность параллельной программы в зависимости от размера матрицы и размера решётки.

# ПРИЛОЖЕНИЕ

#include <iostream>

#include <mpi.h>

#define M 256

#define N 256

#define K 256

using namespace std;

void initMatrix(double\* matrix, int n, int m, double val) {

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < m; j++) {

            matrix[i \* n + j] = val;

        }

    }

}

void mult(int \*matrixSizes, double \*A, double \*B, double \*C, int \*gridSizes, MPI\_Comm comm) {

    double \*submA = nullptr;

    double \*submB = nullptr;

    double \*submC = nullptr;

    int submSizes[2];

    int coords[2];

    int rank;

    int \*sendcountsB = nullptr;

    int \*displsScatterB = nullptr;

    int \*recvcountsGatherC = nullptr;

    int \*displsGatherC = nullptr;

    MPI\_Datatype typeB, typeC, types[2];

    int blockLengths[2] = {1, 1};

    int periods[2] = {0, 0};

    int remainDims[2];

    MPI\_Comm comm2d;

    MPI\_Comm comm1d[2];

    MPI\_Bcast(matrixSizes, 3, MPI\_INT, 0, comm);

    MPI\_Bcast(gridSizes, 2, MPI\_INT, 0, comm);

    MPI\_Cart\_create(comm, 2, gridSizes, periods, false, &comm2d);

    MPI\_Comm\_rank(comm2d, &rank);

MPI\_Cart\_coords(comm2d, rank, 2, coords);

    for (int i = 0; i < 2; i++) {

        for (int j = 0; j < 2; j++) {

            remainDims[j] = (i == j);

        }

        MPI\_Cart\_sub(comm2d, remainDims, &comm1d[i]);

    }

    submSizes[0] = matrixSizes[0] / gridSizes[0];

    submSizes[1] = matrixSizes[2] / gridSizes[1];

    submA = new double[submSizes[0] \* matrixSizes[1]];

    submB = new double[matrixSizes[1] \* submSizes[1]];

    submC = new double[submSizes[0] \* submSizes[1]];

    if (rank == 0) {

        MPI\_Type\_vector(matrixSizes[1], submSizes[1], matrixSizes[2], MPI\_DOUBLE, &types[0]);

        long int size;

        MPI\_Type\_extent(MPI\_DOUBLE, &size);

        types[1] = MPI\_UB;

        long int \*displacements = new long int[2];

        displacements[0] = 0;

        displacements[1] = size \* submSizes[1];

        MPI\_Type\_create\_struct(2, blockLengths, displacements, types, &typeB);

        MPI\_Type\_commit(&typeB);

        displsScatterB = new int[gridSizes[1]];

        sendcountsB = new int[gridSizes[1]];

        for (int i = 0; i < gridSizes[1]; i++) {

            displsScatterB[i] = i;

            sendcountsB[i] = 1;

        }

        MPI\_Type\_vector(submSizes[0], submSizes[1], matrixSizes[2], MPI\_DOUBLE, &typeC);

        MPI\_Type\_create\_struct(2, blockLengths, displacements, types, &typeC);

        MPI\_Type\_commit(&typeC);

        displsGatherC = new int[gridSizes[0] \* gridSizes[1]];

        recvcountsGatherC = new int[gridSizes[0] \* gridSizes[1]];

        for (int i = 0; i < gridSizes[0]; i++) {

            for (int j = 0; j < gridSizes[1]; j++) {

                displsGatherC[i \* gridSizes[1] + j] = (i \* gridSizes[1] \* submSizes[0] + j);

                recvcountsGatherC[i \* gridSizes[1] + j] = 1;

            }

        }

        delete[] displacements;

    }

    if (coords[1] == 0) {

        MPI\_Scatter(A, submSizes[0] \* matrixSizes[1], MPI\_DOUBLE, submA, submSizes[0] \* matrixSizes[1], MPI\_DOUBLE, 0, comm1d[0]);

    }

    if (coords[0] == 0) {

        MPI\_Scatterv(B, sendcountsB, displsScatterB, typeB, submB, submSizes[1] \* matrixSizes[1], MPI\_DOUBLE, 0, comm1d[1]);

    }

    MPI\_Bcast(submA, submSizes[0] \* matrixSizes[1], MPI\_DOUBLE, 0, comm1d[1]);

    MPI\_Bcast(submB, submSizes[1] \* matrixSizes[1], MPI\_DOUBLE, 0, comm1d[0]);

    int m = submSizes[1];

    int n = matrixSizes[1];

    for (int i = 0; i < submSizes[0]; i++) {

        for (int j = 0; j < m; j++) {

            submC[i \* m + j] = 0;

            for (int k = 0; k < n; k++) {

                submC[i \* m + j] += submA[n \* i + k] \* submB[m \* k + j];

            }

        }

    }

    MPI\_Gatherv(submC, submSizes[0] \* submSizes[1], MPI\_DOUBLE, C, recvcountsGatherC, displsGatherC, typeC, 0, comm2d);

    delete[] submA;

    delete[] submB;

    delete[] submC;

    delete[] displsScatterB;

    delete[] sendcountsB;

    MPI\_Comm\_free(&comm2d);

    MPI\_Comm\_free(&comm1d[0]);

    MPI\_Comm\_free(&comm1d[1]);

    if (rank == 0) {

        delete[] recvcountsGatherC;

        delete[] displsGatherC;

        MPI\_Type\_free(&typeB);

        MPI\_Type\_free(&typeC);

        MPI\_Type\_free(&types[0]);

    }

}

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

    double begin, end;

    int size, rank;

    int matrixSizes[3];

    int gridSizes[2];

    double \*A = nullptr;

    double \*B = nullptr;

    double \*C = nullptr;

    MPI\_Init(&argc, &argv);

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

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

    if (argc != 3) {

        if (rank == 0) {

            cout << "Empty grid size"<<endl;

        }

        exit(1);

    }

    int p1 = atoi(argv[1]);

    int p2 = atoi(argv[2]);

    if (p1 == 0 || p2 == 0) {

        if (rank == 0) {

            cout << "Invalid grid size"<<endl;

        }

        exit(1);

    }

    if (size != p1 \* p2) {

        if (rank == 0) {

            cout << "Wrong grid size"<<endl;

        }

        exit(1);

    }

    if (rank == 0) {

        matrixSizes[0] = M;

        matrixSizes[1] = N;

        matrixSizes[2] = K;

        gridSizes[0] = p1;

        gridSizes[1] = p2;

        A = new double[M \* N];

        B = new double[N \* K];

        C = new double[M \* K];

        initMatrix(A, M, N, 1);

        initMatrix(B, N, K, 1);

        initMatrix(C, M, K, 0);

    }

    begin = MPI\_Wtime();

    mult(matrixSizes, A, B, C, gridSizes, MPI\_COMM\_WORLD);

    end = MPI\_Wtime();

    if (rank == 0) {

        cout << "Time taken: " << end - begin << " [s]"<<endl;

        delete[] A;

        delete[] B;

        delete[] C;

    }

    MPI\_Finalize();

    return 0;

}