###### 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 решетка»

студентки 2 курса, группы 23210

**Луньковой Екатерины Евгеньевны**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Артюхов.А.А

Новосибирск 2025

**СОДЕРЖАНИЕ**

[ЦЕЛИ 3](#_Toc114782465)

[ЗАДАНИЕ 3](#_Toc114782466)

[ОПИСАНИЕ РАБОТЫ 4-6](#_Toc114782467)

[ЗАКЛЮЧЕНИЕ 7](#_Toc114782468)

[Приложение 1. *Исходный код программы на “С++”* 8](#_Toc114782469)-11

[Приложение 2. *Замеры времени при разных размерах решетки и разного количества процессов.* 11](#_Toc114782470)

[Приложение 3. *Профилирование программы.* 11](#_Toc114782470)

# ЦЕЛИ

Изучить параллельный алгоритм умножения матрицы на матрицу при 2D решетке.

# 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

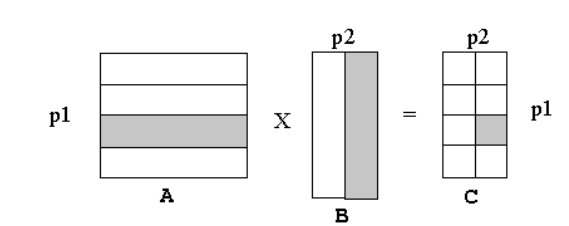
1. Была написана программа на языке С++, реализующая параллельный алгоритм умножения матрицы на матрицу при 2D решетке.

**Алгоритм умножения матрицы на матрицу при 2D решетке:**

* 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 . 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). Выделенные данные расположены в одном компьютере.



*Рис. 1. Разрезание данных для параллельного алгоритма произведения двух матриц при вычислении в 2D решетке компьютеров.*

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:

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

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

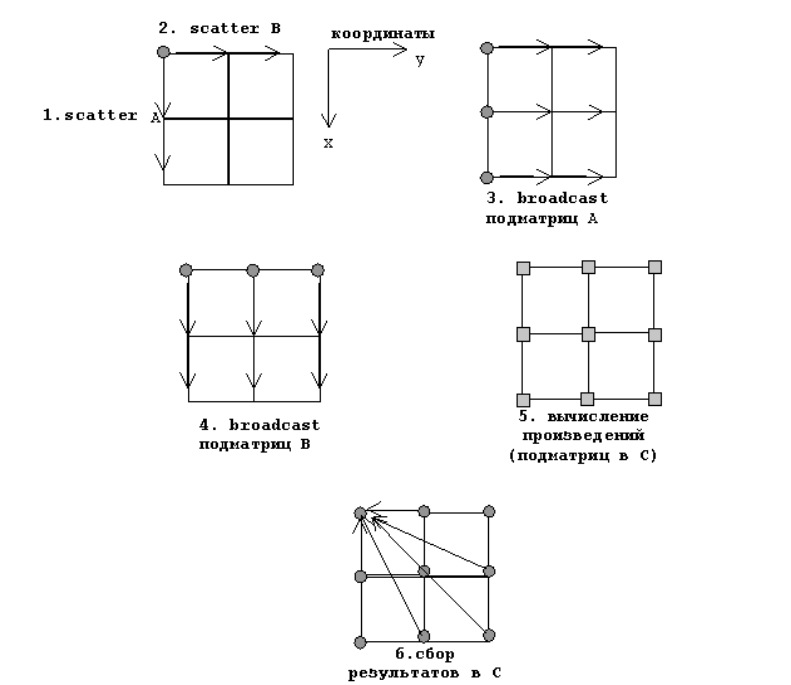
3) Полосы А распространяются в измерении y.

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*

1. Функции, которые использовались в коде:
2. Создание топологии:  
   MPI\_Cart\_create -  создаёт коммуникатор с декартовой топологией (решеткой).

**Параметры:**

* MPI\_COMM\_WORLD - исходный коммуникатор
* 2 - количество измерений решетки
* dims - массив, определяющий количество процессов в каждом измерении
* periods - массив, определяющий, является ли каждое измерение периодическим (1) или нет (0)
* 1 - флаг, указывающий, можно ли переупорядочивать процессы (1 - можно)
* &grid\_comm - выходной параметр для нового коммуникатора

1. Задаются координаты процесса для определения позиции процесса в решетке, а так же для создания коммуникатора для столбцов и строк.

MPI\_Cart\_coords - возвращает координаты процесса в декартовой решетке.

**Параметры**:

* grid\_comm - коммуникатор с декартовой топологией
* rank - ранг процесса в grid\_comm
* 2 - количество измерений
* coords - выходной массив для координат

1. Разделение на две топологии “row\_comm” и “col\_comm” :

MPI\_Comm\_split -  создаёт новые коммуникаторы, разделяя исходный на группы.

**Параметры**:

* grid\_comm - исходный коммуникатор
* color - определяет, к какой группе попадет процесс (одинаковые color - в одну группу)
* key - определяет порядок рангов в новой группе
* &newcomm - выходной параметр для нового коммуникатора.

(key (j для строк и i для столбцов) определяет порядок процессов в новой группе).

1. После инициализации матрицы на 0 процессе по заданию нужно распределить по топологиям.  
   MPI\_Scatter - распределяет данные от корневого процесса всем процессам в коммуникаторе.

**Параметры**:

* A.data() - буфер отправки (только у корневого процесса)
* local\_n1 \* n2 - количество элементов для отправки каждому процессу
* MPI\_INT - тип данных
* local\_A.data() - буфер приема
* 0 - ранг корневого процесса в col\_comm
* col\_comm – коммуникатор

1. После получения данных нужно копию распространить по строкам и столбцам.

# MPI\_Bcast - транслирует данные от одного процесса всем процессам.

**Параметры**:

* A.data() - буфер отправки
* local\_n1 \* n2 n2 - количество элементов для отправки каждому процессу
* MPI\_INT - тип данных
* 0 - ранг процесса, отправляющего данные
* row\_comm - коммуникатор

1. Сделаны замеры времени работы программы при разных размерах решетки и разного количества процессов (приложение 2).

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы был реализован параллельный алгоритм умножения матриц с использованием двумерной декартовой топологии (2D-решетки) в MPI. Программа успешно распределяет вычисления между процессами, обеспечивая эффективное умножение матриц за счет декомпозиции данных и минимизации пересылок.

# Приложение 1.  Исходный код программы на “С++”

#include <iostream>

#include <vector>

#include <mpi.h>

#include <cstdlib>

#include <ctime>

using namespace std;

void initializeMatrices(int rank, vector<int>& A, vector<int>& B, int n1, int n2, int n3) {

if (rank == 0) {

A.resize(n1 \* n2);

B.resize(n2 \* n3);

for (int i = 0; i < n1; ++i) {

for (int j = 0; j < n2; ++j) {

A[i \* n2 + j] = 1 + (i + j) % 3;

}

}

for (int i = 0; i < n2; ++i) {

for (int j = 0; j < n3; ++j) {

B[i \* n3 + j] = 1 + (i + j) % 4; // Пример: B[i][j] = i - j

}

}

}

}

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

MPI\_Init(&argc, &argv);

int rank, size;

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

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

int p1 = 8, p2 = 2;

if (p1 \* p2 != size) {

if (rank == 0) {

cerr << "Ошибка: Размеры сетки не соответствуют количеству процессов" << endl;

}

MPI\_Finalize();

return 1;

}

// Размеры матриц

int n1 = 1024, n2 = 1024, n3 = 512;

if (n1 % p1 != 0 || n3 % p2 != 0) {

if (rank == 0) {

cerr << "Ошибка: Размеры не делятся на сетку" << endl;

}

MPI\_Finalize();

return 1;

}

int local\_n1 = n1 / p1; // Количество строк в локальной матрице A

int local\_n3 = n3 / p2; // Количество столбцов в локальной матрице B

// Инициализация матриц A и B

vector<int> A, B;

initializeMatrices(rank, A, B, n1, n2, n3);

// Создание топологии решетки

int dims[2] = {p1, p2};

int periods[2] = {0, 0};

MPI\_Comm grid\_comm;

MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims, periods, 1, &grid\_comm);

// Получение координат текущего процесса в решетке

int coords[2];

MPI\_Cart\_coords(grid\_comm, rank, 2, coords);

int i = coords[0], j = coords[1];

// Локальные матрицы

vector<int> local\_A(local\_n1 \* n2);

vector<int> local\_B(n2 \* local\_n3);

vector<int> local\_C(local\_n1 \* local\_n3, 0);

MPI\_Comm row\_comm, col\_comm;

MPI\_Comm\_split(grid\_comm, i, j, &row\_comm);

MPI\_Comm\_split(grid\_comm, j, i, &col\_comm);

double start\_time, end\_time;

if (rank == 0) {

start\_time = MPI\_Wtime();

}

if (j == 0) {

MPI\_Scatter(A.data(), local\_n1 \* n2, MPI\_INT,

local\_A.data(), local\_n1 \* n2, MPI\_INT,

0, col\_comm);

}

MPI\_Bcast(local\_A.data(), local\_n1 \* n2, MPI\_INT, 0, row\_comm);

if (i == 0) {

MPI\_Scatter(B.data(), n2 \* local\_n3, MPI\_INT,

local\_B.data(), n2 \* local\_n3, MPI\_INT,

0, row\_comm);

}

MPI\_Bcast(local\_B.data(), n2 \* local\_n3, MPI\_INT, 0, col\_comm);

for (int x = 0; x < local\_n1; ++x) {

for (int y = 0; y < local\_n3; ++y) {

for (int k = 0; k < n2; ++k) {

local\_C[x \* local\_n3 + y] +=

local\_A[x \* n2 + k] \* local\_B[k \* local\_n3 + y];

}

}

}

vector<int> C;

if (rank == 0) {

C.resize(n1 \* n3);

}

MPI\_Gather(local\_C.data(), local\_n1 \* local\_n3, MPI\_INT,

C.data(), local\_n1 \* local\_n3, MPI\_INT,

0, MPI\_COMM\_WORLD);

if (rank == 0) {

end\_time = MPI\_Wtime();

cout << "Time taken: " << end\_time - start\_time << " seconds" << endl;

}

MPI\_Comm\_free(&row\_comm);

MPI\_Comm\_free(&col\_comm);

MPI\_Comm\_free(&grid\_comm);

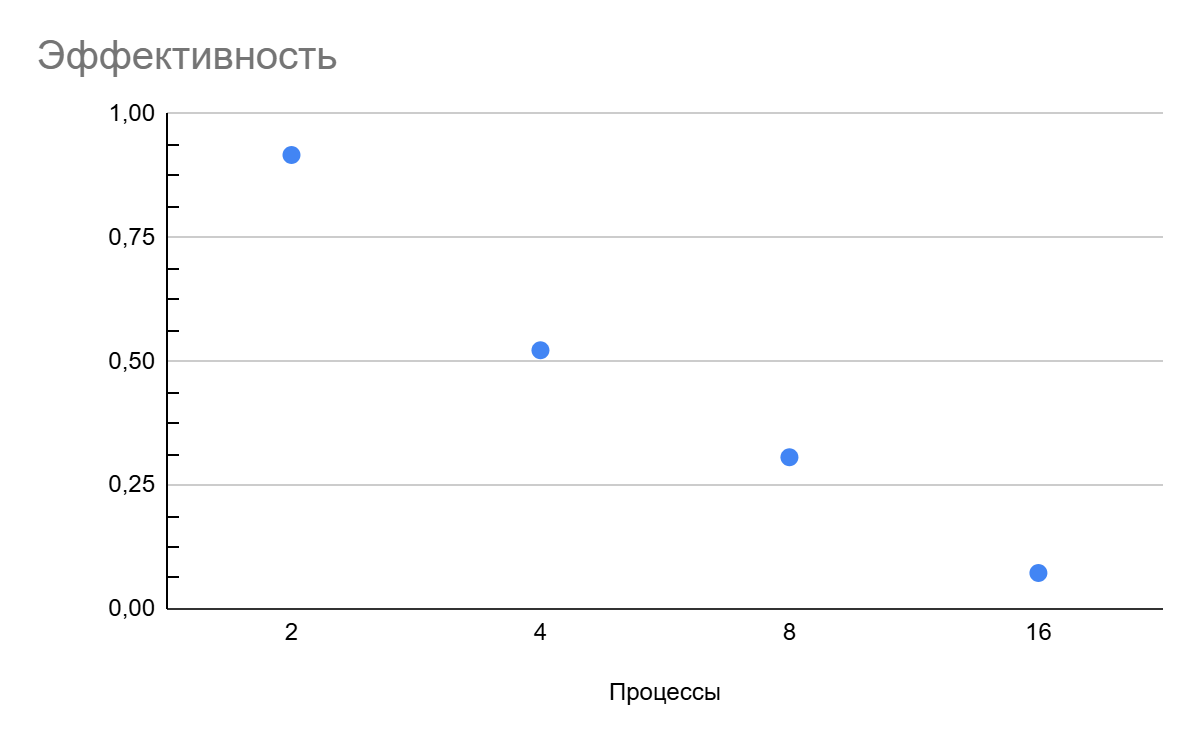
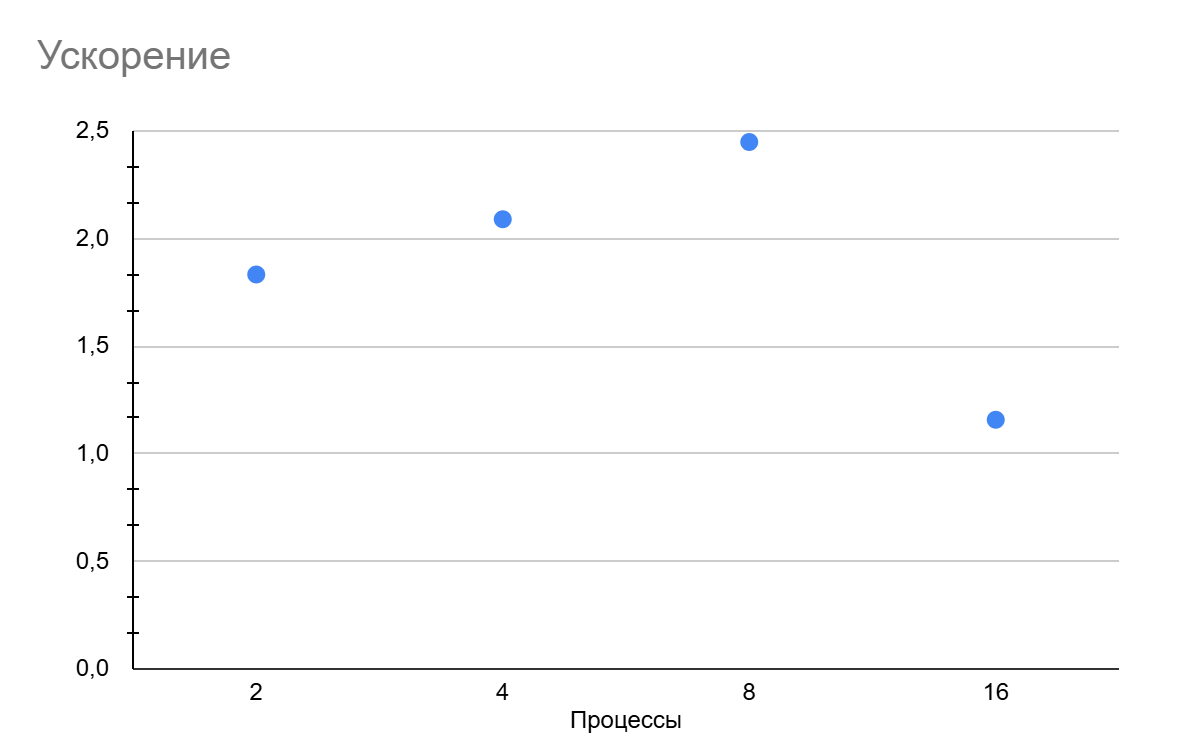
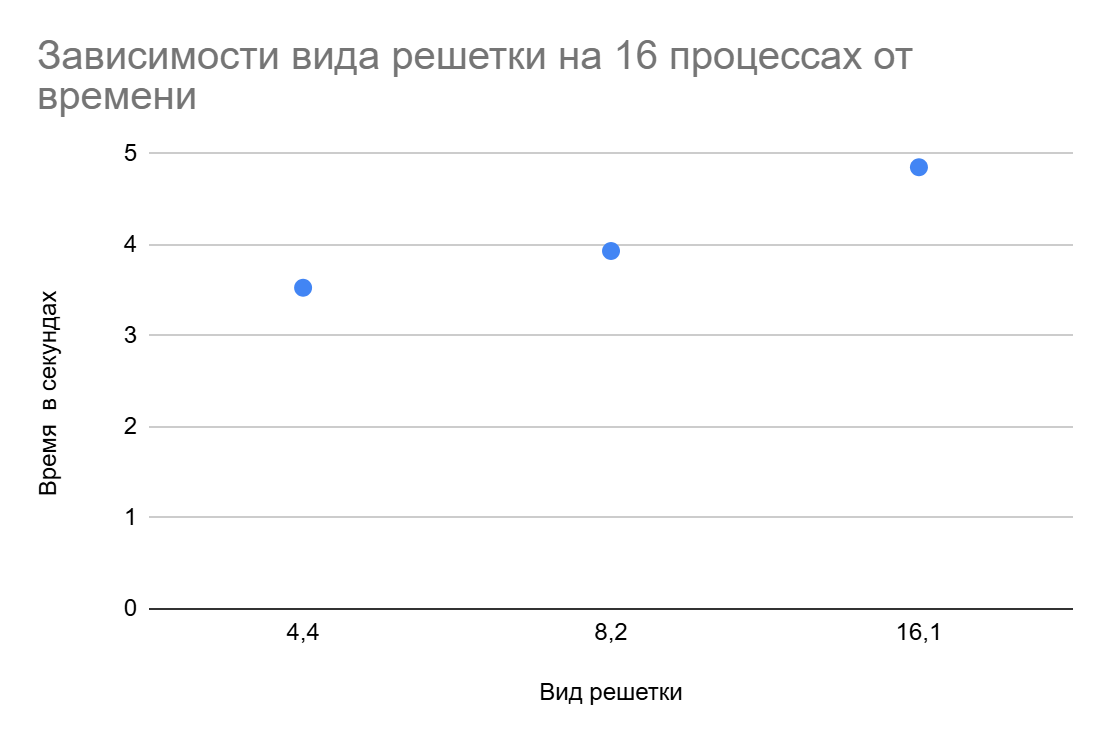
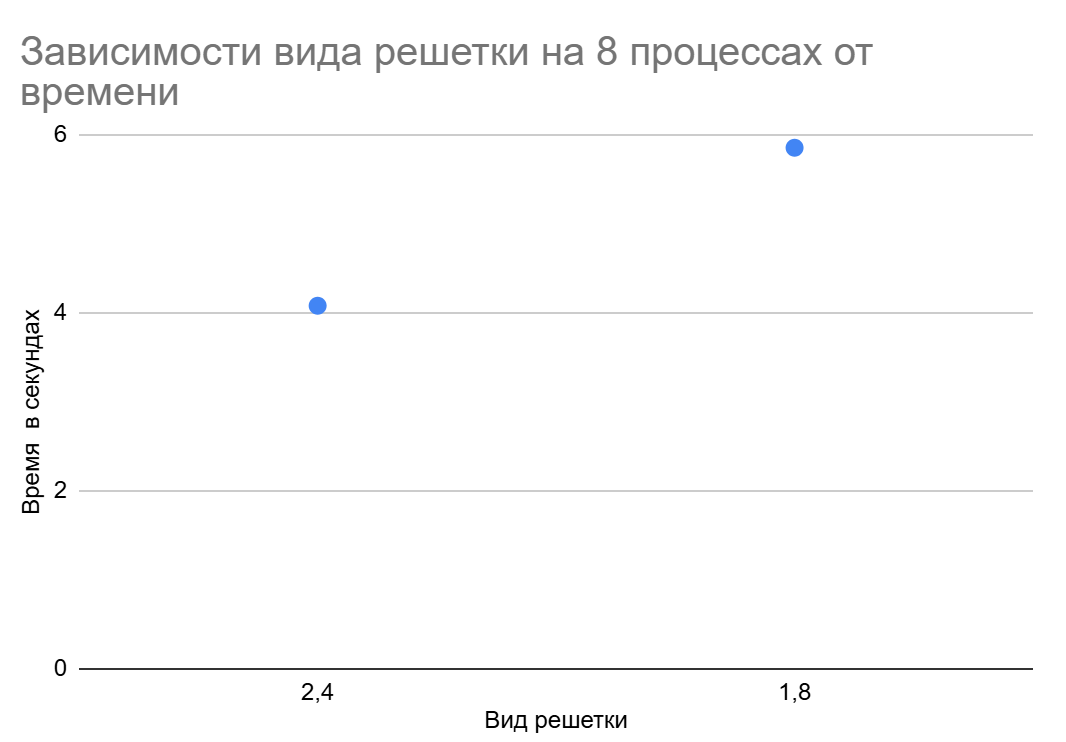
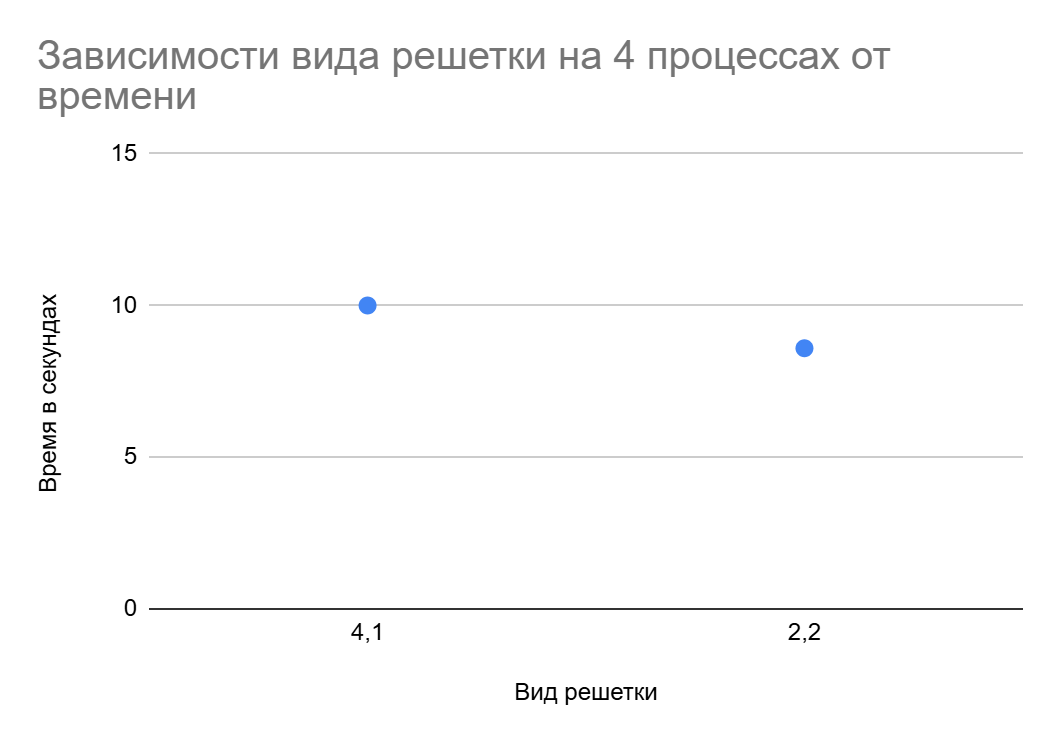
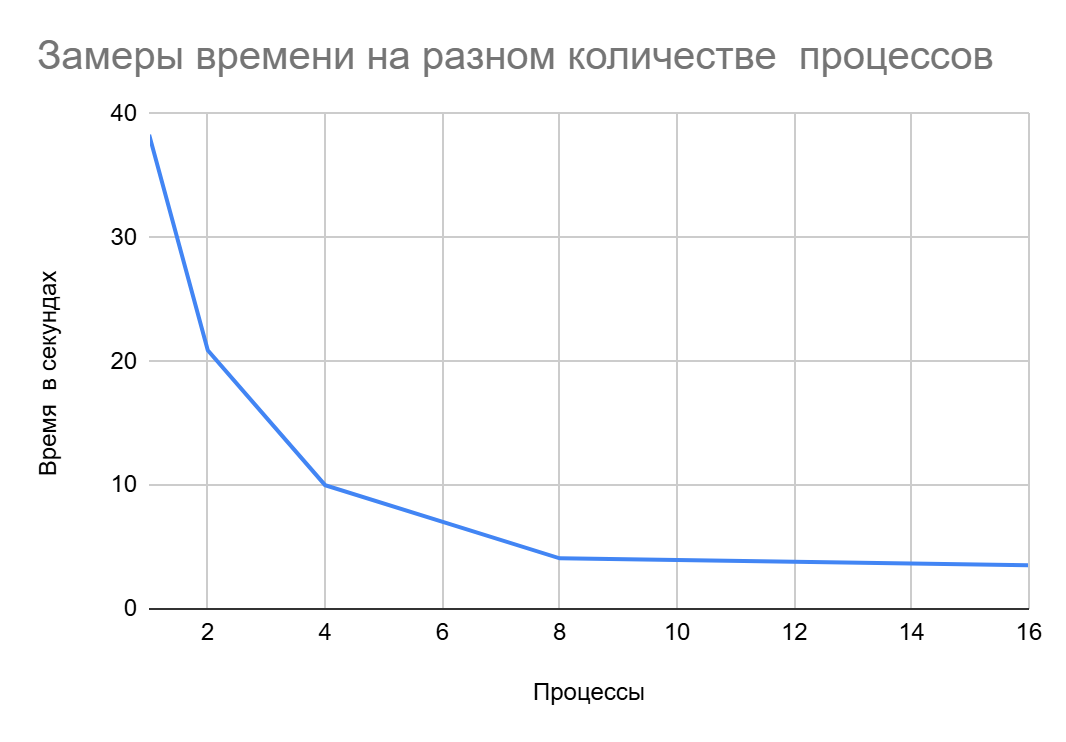
MPI\_Finalize();

return 0;

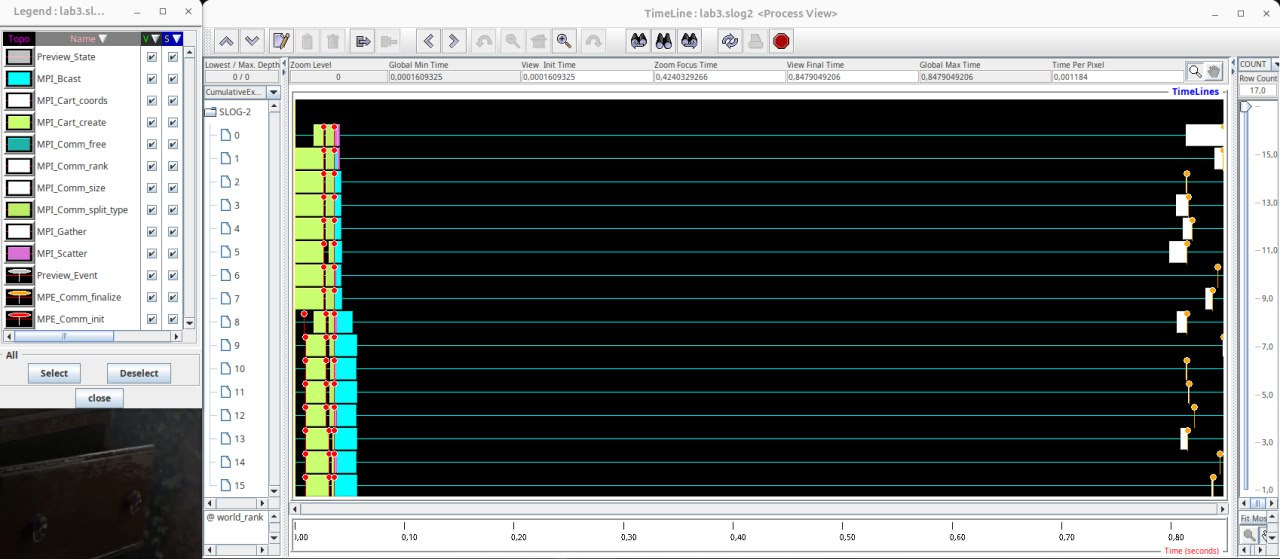
}

# Приложение 2.  Замеры времени при разных размерах решетки и разного количества процессов.

|  |  |  |
| --- | --- | --- |
| Количество процессов | Размер решетки | Время в секундах |
| 1 | (1,1) | 38.3033 |
| 2 | (2.1) | 20.8938 |
| 4 | (4,1) | 9.99729 |
|  | (2,2) | 8.59364 |
| 8 | (2,4) | 4.08289 |
|  | (1,8) | 5.86029 |
| 16 | (4,4) | 3.52757 |
|  | (8.2) | 3.93354 |
|  | (16,1) | 4.85243 |



# Приложение 3.  Профилирование программы



На этом профилировании видно, что данные распределяется через одного, это связано с тем, что в коде я использую функцию Scatter, которая распределяет уникальную часть матрицы по коммуникаторам.