Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Пермский национальный исследовательский

политехнический университет»

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

по дисциплине **«Технологии блокчейн и распределенные информационные системы»**

**Выполнил** студент гр. АСУ-20-1б

Чувашев Максим

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Проверил** доцент кафедры ИТАС

Щапов Владислав Алексеевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(оценка, дата, подпись преподавателя)*

Пермь, 2024

# 1 Анализ предметной области

## 1.1 Постановка задачи

С использованием языка C++ и библиотеки OpenMPI разработать программу для решения умножения матриц.

## 1.2 Инструменты разработки

Для выполнения лабораторной работы были использованы следующие инструменты:

1. редактор CLion;

# 2 Технология разработки

Матрицы «A» и «B» генерируются случайным образом с заданным размером «MATRIX\_SIZE». В основном 0 процессе генерируются матрицы и выводятся в консоль. Далее с помощью «MPI\_Bcast» матрицы из 0 процесса рассылаются во все процессы. Данная часть представлена на листинге 1.

if (rank == 0) {

generateRandomMatrix(A);

generateRandomMatrix(B);

cout << "Matrix A:" << endl;

printMatrix(A);

cout << "Matrix B:" << endl;

printMatrix(B);

}

MPI\_Bcast(A, MATRIX\_SIZE \* MATRIX\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(B, MATRIX\_SIZE \* MATRIX\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

Листинг 1 — Распределение данных по процессам

Далее вычисляется размер блока С, который будет иметь каждый поток. Далее матрица А умножается на матрицу B по частям. В каждом потоке используется своя часть матрицы A. И результат умножения будет заноситься в локальную матрицу C. Далее данные из всех локальных матриц C собираются с помощью «MPI\_Gather» в большую матрицу C.

int blockSize = MATRIX\_SIZE / size;

int localC[blockSize][MATRIX\_SIZE];

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

for (int j = 0; j < MATRIX\_SIZE; ++j) {

localC[i][j] = 0;

for (int k = 0; k < MATRIX\_SIZE; ++k) {

localC[i][j] += A[i + rank \* blockSize][k] \* B[k][j];

}

}

}

MPI\_Gather(localC, blockSize \* MATRIX\_SIZE, MPI\_INT, C, blockSize \* MATRIX\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

Листинг 2 — Умножение матриц

# 3 Результаты работы

Результаты выполнения задания лабораторной работы проиллюстрированы на рисунке 1.

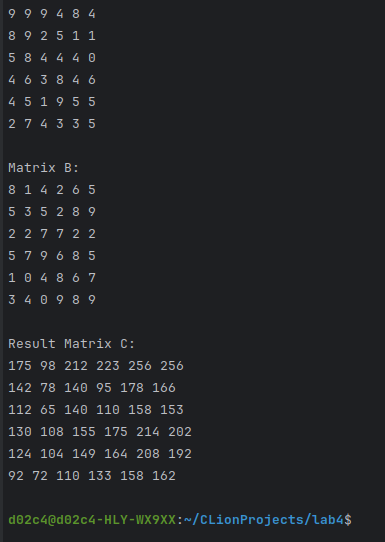


Рисунок 1 — Результат работы программы

ПРИЛОЖЕНИЕ А

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <mpi.h>

using namespace std;

const int MATRIX\_SIZE = 6;

void generateRandomMatrix(int matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = 0; i < MATRIX\_SIZE; ++i) {

for (int j = 0; j < MATRIX\_SIZE; ++j) {

matrix[i][j] = rand() % 10;

}

}

}

void printMatrix(int matrix[MATRIX\_SIZE][MATRIX\_SIZE]) {

for (int i = 0; i < MATRIX\_SIZE; ++i) {

for (int j = 0; j < MATRIX\_SIZE; ++j) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

cout << endl;

}

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

int rank, size;

MPI\_Init(&argc, &argv);

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

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

int A[MATRIX\_SIZE][MATRIX\_SIZE], B[MATRIX\_SIZE][MATRIX\_SIZE], C[MATRIX\_SIZE][MATRIX\_SIZE];

srand(time(NULL) + rank);

if (rank == 0) {

generateRandomMatrix(A);

generateRandomMatrix(B);

cout << "Matrix A:" << endl;

printMatrix(A);

cout << "Matrix B:" << endl;

printMatrix(B);

}

MPI\_Bcast(A, MATRIX\_SIZE \* MATRIX\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(B, MATRIX\_SIZE \* MATRIX\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

int blockSize = MATRIX\_SIZE / size;

int localC[blockSize][MATRIX\_SIZE];

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

for (int j = 0; j < MATRIX\_SIZE; ++j) {

localC[i][j] = 0;

for (int k = 0; k < MATRIX\_SIZE; ++k) {

localC[i][j] += A[i + rank \* blockSize][k] \* B[k][j];

}

}

}

MPI\_Gather(localC, blockSize \* MATRIX\_SIZE, MPI\_INT, C, blockSize \* MATRIX\_SIZE, MPI\_INT, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

cout << "Result Matrix C:" << endl;

printMatrix(C);

}

MPI\_Finalize();

return 0;

}