Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

 «Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Технологии блокчейн и распределенные информационные системы»

Профиль: «Разработка программно-информационных систем»

ОТЧЕТ

по лабораторной работе №4

Тема: «Умножение матриц с использованием MPI»

Выполнил: студент группы РИС-20-2б

Катаев М.М.   \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель кафедры ИТАС

Щапов В. А. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2024

**Цели и задачи**

1. Реализовать программу для умножения матриц с использованием MPI

2. Проанализировать результаты

3. Сделать выводы

**Выполнение работы**

MPI - это стандарт на программный инструментарий для обеспечения связи между ветвями параллельного приложения.

Первой была написана функция для вычисления произведения матриц.

Получение общего количества параллельных задач и порядковый номер вызывающей задачи происходит с помощью **MPI\_Comm\_size()** и

**MPI\_Comm\_rank()**.

**MPI\_Scatter** части распределяются по приемным буферам всех задач, в нашем случае мы распределяем строки матрицы **A**.

**MPI\_Bcast** транслирует данные от одного участника группы всем участникам группы, например, матрицу **B**.

**MPI\_Barrier(MPI\_COMM\_WORLD)** выполняет роль синхронизатора, то есть пока все задачи не выполнятся, ни одна задача не продолжит свое выполнение после барьера.

**MPI\_Gather** противоположность MPI\_Scatter, собирает с каждой задачи части и формирует финальную матрицу **C** в родительской задаче.

Логика работы программы:

1. Программа инициализирует MPI, определяет номер текущего процесса и общее количество процессов.
2. Матрица a разбивается на части и рассылается по процессам с помощью MPI\_Scatter.
3. Матрица **B** транслируется всем процессам с помощью MPI\_Bcast.
4. Каждый процесс вычисляет свою часть произведения матриц и сохраняет результат в массив **localC**.
5. Процессы синхронизируются с помощью MPI\_Barrier.
6. Результаты сбора **localC** от всех процессов собираются на процессе 0 с помощью MPI\_Gather.
7. Процесс 0 выводит полученную матрицу.

Ниже приведет полный код программы.

Листинг 1 – Код программы

﻿#include "mpi.h"

#include <ctime>

#include <iostream>

using namespace std;

const int N = 10;

int B[N][N], A[N][N];

int C[N][N];

void setMatrix(int matrix[N][N]) {

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

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

matrix[i][j] = rand() % 10 - 1;

}

}

}

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

{

setlocale(LC\_ALL, "RU");

int i, j, k, rank, size, sum = 0;

int localA[N], localC[N];

MPI\_Init(&argc, &argv);

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

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

if (rank == 0)

{

setMatrix(A);

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

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

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

}

cout << endl;

}

setMatrix(B);

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

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

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

}

cout << endl;

}

}

MPI\_Scatter(A, N \* N / size, MPI\_INT, localA, N \* N / size, MPI\_INT, 0, MPI\_COMM\_WORLD);//Распределяем каждому процессу по одной строке матрицы

MPI\_Bcast(B, N \* N, MPI\_INT, 0, MPI\_COMM\_WORLD);//Распределяем каждому процессу по одной строке матрицы

for (i = 0; i < N; i++)

{

for (j = 0; j < N; j++)

{

sum = sum + localA[j] \* B[j][i];

//cout << " localA[" << j << "] " << localA[j] << " \* B[" << j << "][" << i << "]" <<B[j][i]<<endl;

}

//cout << "rank: " << rank << " i: " << i << " sum: " << sum <<endl ;

localC[i] = sum;

sum = 0;

}

cout<<"proc "<< rank <<" data: ";//Номер процесса и данные с которыми он работает (localA)

for (i = 0; i < N ; i++) {

cout<<localA[i]<<" ";

}

cout<<"\n";

MPI\_Barrier(MPI\_COMM\_WORLD);//Ождиаем пока все процессы выполнят код до текущей строки для синхронизации

MPI\_Gather(localC, N \* N / size, MPI\_INT, C, N \* N / size, MPI\_INT, 0, MPI\_COMM\_WORLD);//Сбор данных из каждого процесса в финальную матрицу C

if (rank == 0)//Вывод финальной матрицы после синхронизации и сбора данных

{

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

for (i = 0; i < N; i++) {

for (j = 0; j < N; j++) {

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

}

cout << endl;

}

cout << endl << endl;

}

MPI\_Finalize();

}

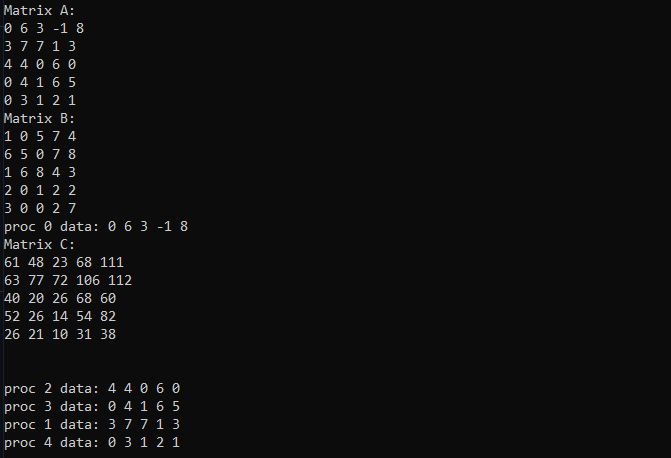


Рисунок 1- результат выполнения

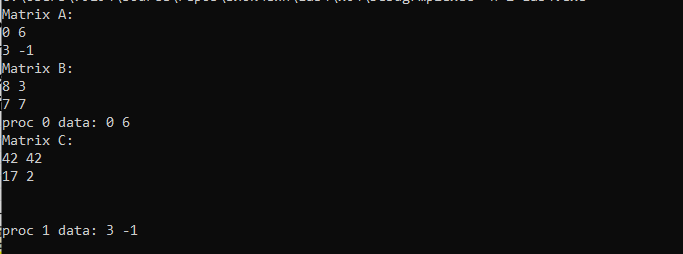


Рисунок 2- результат выполнения

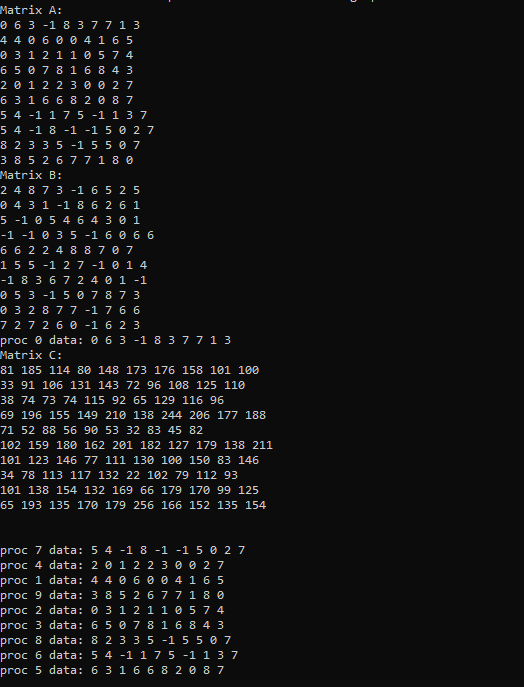


Рисунок 3- результат выполнения

На рисунках видно, что каждый процесс забрал на себя определенную строку(localA) матрицы **A**,тем самым позволил выполнить эту часть кода параллельно и ускорить её выполнение.