Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Теория распределенных систем и параллельных вычислений”

Лабораторная работа №2

“Исследование коллективного типа передачи данных,

групп и коммуникаторов в MPI”

Выполнил:

ст.гр. ИС/б-17-2

Долженко И.А.

Проверил:

Дрозин А.Ю.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать способы обмена данными между процессами в режиме широковещания или группового обмена с использованием MPI-функций.

2 ПОСТАНОВКА ЗАДАЧИ

**Вариант №1.** Реализовать блочный алгоритм распределенного параллельного перемножения матриц Aи Bс размерами (8\*5) и (5\*3) соответственно. Вид распределяемых между процессами блоков представлен на рисунке 1:



Рисунок 1 – Перемножение матриц

Корневой процесс реализует рассылку:

1) блоков матрицы A по две строки;

2) широковещательную рассылку элементов матрицы B между обрабатывающими процессами внутри своей группы (по умолчанию) – режим **One-To-All**.

Организуется четыре процесса, обрабатывающих данные и формирующих фрагменты (2\*3) матрицы результата C. После подготовки блоков матрицы со всеми обрабатывающими процессами (взаимная синхронизация функцией **MPI\_Barrier**) выполняется совместная передача результатов (блоков матрицы C) корневому процессу – режим **All-To-One**.

3 КОД ПРОГРАММЫ

#include <iostream>

#include <fstream>

#include "mpi.h"

using namespace std;

double\*\* InitMatrix(int row, int column);

double\*\* ReadMatrixFromFile(string path, int\* row, int\* column);

void PrintMatrix(double\*\* matrix, int row, int column);

double\* fromMatrixToVector(double\*\* matrix, int row, int column);

double\*\* fromVectorToMatrix(double\* vector, int row, int column);

double\* MatrixMultyply(double\*\* partMatrixA, int rowA, int columnA, double\*\* matrixB, int rowB, int columnB);

double\* MultiplyRowsByColumn(double\* vectorA, double\*\* matrixB, int vector\_column, int matrix\_column);

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

{

int rowA = 8, columnA = 5, rowB = 5, columnB = 3, buffSize = 5 \* 3;

MPI\_Status status;

int rank, size, i = 0, k = 2;

double\*\* matrixA, \*\* matrixB, \* buffer = new double[buffSize];

double\* mainResult = new double[rowA \* columnB];

double\* partResult = new double[k \* columnB];

double\* vectorToSend = new double[(rowA / k) \* columnA \* k];

double\* recvBuffer = new double[k \* columnA];

MPI\_Init(&argc, &argv);

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

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

if (rank == 0) {

// 1) рассылаем matrixA по 2 строки по процессам

matrixA = ReadMatrixFromFile("matrixA.txt", &rowA, &columnA);

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

PrintMatrix(matrixA, rowA, columnA);

// создаем из каждых двух строк матрицы один вектор

double\*\* matrixToSender = InitMatrix(rowA / k, columnA \* k);

for (int i = 0, number = 0; i < rowA; number++) {

double\*\* temp = InitMatrix(k, columnA);

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

temp[j] = matrixA[i++];

}

matrixToSender[number] = fromMatrixToVector(temp, k, columnA);

}

vectorToSend = fromMatrixToVector(matrixToSender, rowA / k, columnA \* k);

// создаем из матрицы matrixB вектор buffer

matrixB = ReadMatrixFromFile("matrixB.txt", &rowB, &columnB);

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

PrintMatrix(matrixB, rowB, columnB);

buffer = fromMatrixToVector(matrixB, rowB, columnB);

}

// отправляем каждому процессу свою часть матрицы А в виде вектора

MPI\_Scatter(vectorToSend, k \* columnA, MPI\_DOUBLE, recvBuffer, k \* columnA, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

// 2) рассылаем matrixB по всем процессам

MPI\_Bcast(buffer, buffSize, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

// 3) в каждом процессе получаем свою часть матрицы А и всю матрицу В в виде векторов

double\*\* partMatrixA = fromVectorToMatrix(recvBuffer, k, columnA);

matrixB = fromVectorToMatrix(buffer, rowB, columnB);

partResult = MatrixMultyply(partMatrixA, k, columnA, matrixB, rowB, columnB);

cout << "Result of matrix multiplication: " << endl;

PrintMatrix(fromVectorToMatrix(partResult, k, columnB), k, columnB);

// 4) собираем результат умножения со всех процессов

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Gather(partResult, k \* columnB, MPI\_DOUBLE, mainResult, k \* columnB, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

cout << "Result: " << endl;

PrintMatrix(fromVectorToMatrix(mainResult, rowA, columnB), rowA, columnB);

}

MPI\_Finalize();

return 0;

}

double\*\* InitMatrix(int row, int column) {

double\*\* matrix = new double\* [row];

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

matrix[i] = new double[column];

}

return matrix;

}

double\*\* ReadMatrixFromFile(string path, int\* row, int\* column) {

ifstream in(path);

double\*\* matrix;

if (in.is\_open()) {

in >> \*row >> \*column;

matrix = InitMatrix(\*row, \*column);

for (int i = 0; i < \*row; i++) {

for (int j = 0; j < \*column; j++) {

in >> matrix[i][j];

}

}

in.close();

return matrix;

}

else {

cout << "Файл не найден!";

return 0;

}

}

void PrintMatrix(double\*\* matrix, int row, int column) {

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

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

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

}

cout << endl;

}

cout << endl;

}

double\* fromMatrixToVector(double\*\* matrix, int row, int column) {

double\* buffer = new double[row \* column];

int k = 0;

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

for (int j = 0; j < column; j++, k++) {

buffer[k] = matrix[i][j];

}

}

return buffer;

}

double\*\* fromVectorToMatrix(double\* vector, int row, int column) {

double\*\* result\_matrix = InitMatrix(row, column);

int index = 0;

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

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

result\_matrix[i][j] = vector[index++];

}

}

return result\_matrix;

}

double\* MatrixMultyply(double\*\* partMatrixA, int rowA, int columnA, double\*\* matrixB, int rowB, int columnB) {

double\*\* result\_multiplication = InitMatrix(rowA, columnB);

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

result\_multiplication[i] = MultiplyRowsByColumn(partMatrixA[i], matrixB, columnA, columnB);

}

return fromMatrixToVector(result\_multiplication, rowA, columnB);

}

double\* MultiplyRowsByColumn(double\* vectorA, double\*\* matrixB, int vector\_column, int matrix\_column) {

double\* result\_vector = new double[vector\_column] {0};

for (int i = 0; i < matrix\_column; i++) {

for (int t = 0; t < vector\_column; t++) {

result\_vector[i] += vectorA[t] \* matrixB[t][i];

}

}

return result\_vector;

}

4 РЕЗУЛЬТАТЫ

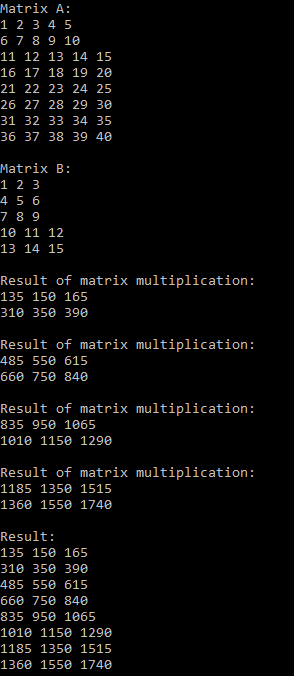


Рисунок 2 – Результат работы программы

ВЫВОДЫ

В ходе выполнения лабораторной работы были исследованы способы обмена данными между процессами в режиме широковещания или группового обмена с использованием MPI-функций. Был реализован блочный алгоритм распределенного параллельного перемножения матриц A и B с размерами (8\*5) и (5\*3) соответственно.