Лабораторная работа №1

ИСЛЕДОВАНИЕ СРЕДСТВ СОЗДАНИЯ РАСПРЕДЕЛЕННО

ВЫПОЛНЯЮЩИХСЯ ПОГРАММ

Цель работы

Исследовать функции библиотеки MPI, необходимые для создания и взаимодействия распределено выполняемых программ.

Постановка задачи

**Вариант №2:** Программа осуществляет вычисление определителя матрицы 4\*4 методом треугольников. Каждый процесс подсчитывает только произве- дения, определение результата осуществляется в родительской задаче, куда передаются результаты работы процесса. По процессам распределяется вся матрица.

Ход работы

Для реализации поставленной задачи была реализована программа на языке С++. Листинг программы представлен ниже.

Листинг 1 – Task.cpp

#include <mpi.h>

#include <iostream>

#include <string>

#include <stdio.h>

#include <cmath>

#include <fstream>

using namespace std;

int \*\*readMatrixFromFile(string filename, int size);

int \*toVector(int \*\*matrix, int size);

int \*\*allocate2DArray(int size);

void freeArray(int \*\*matrix, int size);

void displayMatrix(int size, int \*\*matrix);

int countVectorDet(int \*v);

int \*\*formTriMatrix(int row, int col, int \*\*matr);

int defaultMatrixSize = 4;

int minorMatrixSize = 3;

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

{

int rank, size;

int determinant = 0;

int \*\*matrix = readMatrixFromFile("data.txt", defaultMatrixSize);

int \*\*triangleMatrix = allocate2DArray(3);

MPI\_Status status;

MPI\_Init(&argc, &argv);

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

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

if (rank == 0)

{

cout << endl

<< "Initial matrix: " << endl

<< endl;

displayMatrix(defaultMatrixSize, matrix);

for (int i = 0; i < defaultMatrixSize; i++)

{

triangleMatrix = formTriMatrix(0, i, matrix);

int \*vector = toVector(triangleMatrix, minorMatrixSize);

MPI\_Send(vector, (minorMatrixSize \* minorMatrixSize), MPI\_INT, (i + 1), 99, MPI\_COMM\_WORLD);

freeArray(triangleMatrix, minorMatrixSize);

delete[] vector;

}

int result;

for (int i = 1; i < size; i++)

{

MPI\_Recv(&result, 1, MPI\_INT, i, 99, MPI\_COMM\_WORLD, &status);

determinant += result;

}

cout << endl

<< "Determinant is: " << determinant << endl;

}

else

{

int \*vector = new int[minorMatrixSize \* minorMatrixSize];

MPI\_Recv(vector, (minorMatrixSize \* minorMatrixSize), MPI\_INT, 0, 99, MPI\_COMM\_WORLD, &status);

int det = countVectorDet(vector);

int answer = matrix[0][rank - 1] \* pow(-1, rank + 1) \* det;

MPI\_Send(&answer, 1, MPI\_INT, 0, 99, MPI\_COMM\_WORLD);

delete[] vector;

cout << "Process rank " << rank << " determinant = " << matrix[0][rank - 1] << " \* (-1)\*\*" << rank + 1 << " \* " << det << " = " << answer << endl;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Finalize();

return 0;

}

int \*\*formTriMatrix(int row, int col, int \*\*matr)

{

int \*\*resultMatrix = new int \*[3];

for (int i = 0; i < minorMatrixSize; i++)

{

resultMatrix[i] = new int[minorMatrixSize];

}

int currentRow = 0;

int currentCol = 0;

for (int i = 0; i < defaultMatrixSize; i++)

{

for (int j = 0; j < defaultMatrixSize; j++)

{

if (i != row && j != col)

{

resultMatrix[currentRow][currentCol] = matr[i][j];

currentCol++;

if (currentCol == minorMatrixSize && currentRow != minorMatrixSize)

{

currentCol = 0;

currentRow++;

}

}

}

}

return resultMatrix;

}

int countVectorDet(int \*v)

{

int directTriangle = v[0] \* v[4] \* v[8] + v[1] \* v[5] \* v[6] + v[2] \* v[3] \* v[7];

int reverseTriangle = v[2] \* v[4] \* v[6] + v[0] \* v[5] \* v[7] + v[1] \* v[3] \* v[8];

return directTriangle - reverseTriangle;

}

void displayMatrix(int size, int \*\*matrix)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

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

}

cout << endl;

}

cout << endl;

}

void freeArray(int \*\*matrix, int size)

{

for (int i = 0; i < size; i++)

{

delete[] matrix[i];

}

delete[] matrix;

}

int \*\*allocate2DArray(int size)

{

int \*\*a = new int \*[size];

for (int i = 0; i < size; i++)

{

a[i] = new int[size];

}

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

a[i][j] = 0;

}

}

return a;

}

int \*toVector(int \*\*matrix, int size)

{

int \*v = new int[size \* size];

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

v[i \* size + j] = matrix[i][j];

}

}

return v;

}

int \*\*readMatrixFromFile(string filename, int size)

{

int \*\*a = allocate2DArray(defaultMatrixSize);

ifstream fp(filename);

if (!fp)

{

cout << "Error, file couldn't be opened" << endl;

exit(1);

}

for (int row = 0; row < size; row++)

{

for (int col = 0; col < size; col++)

{

fp >> a[row][col];

if (!fp)

{

cout << "Error reading file for element " << row << "," << col << endl;

exit(1);

}

}

}

fp.close();

return a;

}

Результаты выполнения программы

Проверим работу программы на следующей матрице 4х4:

8 -8 8 4

-6 -8 2 -8

5 4 -2 -3

9 6 -5 9

На рисунке 1 представлен результат выполнения программы.

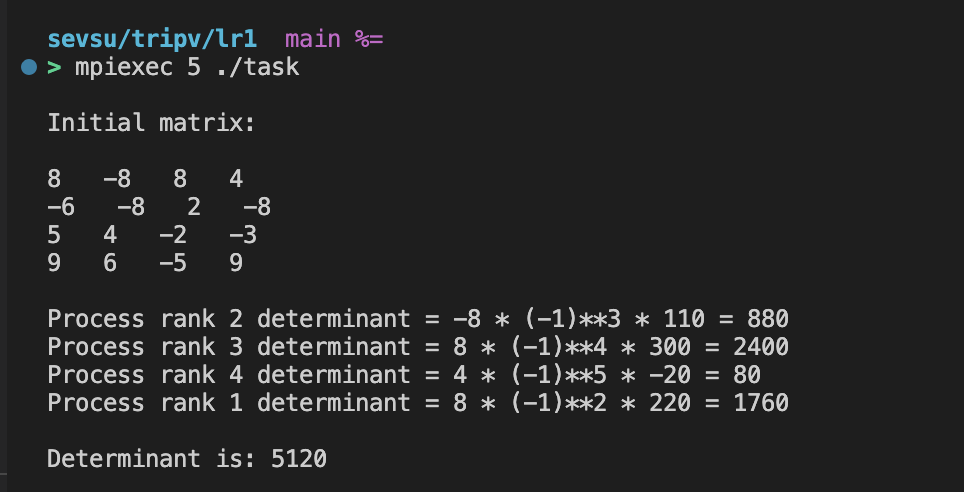


Рисунок 1 - Результат выполнения программы

Если посчитать определитель вручную, то можно обнаружить, что результаты расчётов совпадают с результатами работы программы, что означает, что программа работает исправно.

Выводы

В ходе лабораторной работы были исследованы функции библиотеки MPI, необходимых для создания и взаимодействия распределённо исполняемых программ. Была создана программа, которая осуществляет вычисление определителя матрицы 4\*4 методом треугольников. Каждый процесс подсчитывает только произведения, определение результата осуществляется в родительской задаче, куда передаются результаты работы процесса. По процессам распределяется вся матрица.