**ЛАБОРАТОРНАЯ РАБОТА №2**

**ИССЛЕДОВАНИЕ КОЛЛЕКТИВНОГО ТИПА ПЕРЕДАЧИ**

**ДАННЫХ, ГРУПП И КОММУНИКАТОРОВ В MPI**

**Цель работы**

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

**Постановка задачи**

**Вариант 2.**

Требуется выполнить вычисление максимального и минимального значения функции *f(x,y)* внутри некоторой области. Функция *f(x,y)* задана в виде: *f(x,y)=sin(x)+ey* , а интервалы изменения параметров функции (ее аргументов) определены следующим образом: x,y принадлежит [0,1]. Интервал дискретизации для каждого из аргументов – 0,1. Тогда по каждому из аргументов получено по 10 значений *f(x,y)*. Полученные значения функции *f(x,y)* сведены в корневом процессе (**root**) в матрицу *А* (количество элементов в матрице А – 100). В программе должны быть реализованы две группы процессов: первая группа процессов выполняет определение минимального значения, вторая группа – максимального, корень первой и второй группы является один и тот же процесс (**root**), в котором выполняется расчет значений функции *f(x,y)* внутри области – формирование матрицы *А*. Для каждой из групп создается свой коммуникатор. Матрица *А* разбита на блоки по 4 элемента (2 строки, 2 столбца) в виде, указанном на рисунке 2.2 (здесь *А*i,j соответствующая подматрица (блок), передаваемая корневым процессом (*i,j)*-ому процессу в одной и другой группах).

Каждый из процессов в группе получает (в результате обмена с корневым процессом) свою подматрицу (блок) и ожидает взаимной синхронизации с другими процессами (функция **MPI\_Barrier**). После чего каждым из процессов в группе вызывается функция, определяющая минимум (максимум) среди элементов подматрицы (блока) – выполнение вычислений с каждым из блоков реализуется параллельно с вычислениями для других блоков. Далее процессом из группы (для соответствующего блока матрицы *А*i,j ) выполняется определение глобального минимума (максимума) внутри каждой их групп (вызов функции **MPI\_Reduce**), после чего глобальные значения **min** и **max** записываются в соответствующие переменные корневого процесса с последующим выводом результатов.

**Ход работы**

Была составлена программа, выполняющая поставленную задачу.

Листинг 1 – lr2.cpp

*#include* <mpi.h>

*#include* <iostream>

*#include* <stdio.h>

*#include* <cmath>

*#define* PI 3.14159265

using namespace std;

double \*\*allocate2DArray(int size);

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

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

double getMin(int size, double \*vector);

double getMax(int size, double \*vector);

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

{

int rank, size;

const int ROOT = 0;

int matrSize = 10;

int blockSize = matrSize \* matrSize / 25;

double globalBlocks[5][25];

double globalMin;

double globalMax;

double localVector[25];

double localMin;

double localMax;

MPI\_Status status;

int errCode;

*if* ((errCode = MPI\_Init(&argc, &argv)) != 0)

{

*return* errCode;

}

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

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

int min\_ranks[5];

int max\_ranks[5];

int min\_c = 0, max\_c = 0;

MPI\_Comm min\_comm, max\_comm;

MPI\_Group group\_world, min\_group, max\_group;

MPI\_Comm\_group(MPI\_COMM\_WORLD, &group\_world);

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

{

*if* (i == ROOT)

{

min\_ranks[min\_c] = i;

max\_ranks[max\_c] = i;

min\_c++;

max\_c++;

}

*if* ((i % 2 != 0) && (i != ROOT))

{

min\_ranks[min\_c] = i;

min\_c++;

}

*if* ((i % 2 == 0) && (i != ROOT))

{

max\_ranks[max\_c] = i;

max\_c++;

}

}

MPI\_Group\_incl(group\_world, 5, min\_ranks, &min\_group);

MPI\_Group\_incl(group\_world, 5, max\_ranks, &max\_group);

MPI\_Comm\_create(MPI\_COMM\_WORLD, min\_group, &min\_comm);

MPI\_Comm\_create(MPI\_COMM\_WORLD, max\_group, &max\_comm);

int subrank, subsize;

*if* (rank == ROOT)

{

double \*\*A = allocate2DArray(matrSize);

*for* (int i = 0; i < matrSize; i++)

{

*for* (int j = 0; j < matrSize; j++)

{

double x = i / 10.0;

double y = j / 10.0;

A[i][j] = sin(x \* PI / 180.0) + exp(y);

}

}

displayMatrix(matrSize, A);

int count = 0;

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

{

*if* ((i % 2 == 0) && (i == 0))

{

*for* (int j = 0; j < matrSize / 2; j++)

{

*for* (int k = 0; k < matrSize / 2; k++)

{

globalBlocks[i + 1][j \* (matrSize / 2) + k] = A[j][k];

}

}

}

*if* ((i % 2 == 0) && (i == 2))

{

*for* (int j = matrSize / 2; j < matrSize; j++)

{

*for* (int k = 0; k < matrSize / 2; k++)

{

globalBlocks[i + 1][(j - (matrSize / 2)) \* (matrSize / 2) + k] = A[j][k];

}

}

}

*if* ((i % 2 != 0) && (i == 1))

{

*for* (int j = 0; j < matrSize / 2; j++)

{

*for* (int k = matrSize / 2; k < matrSize; k++)

{

globalBlocks[i + 1][j \* (matrSize / 2) + (k - (matrSize / 2))] = A[j][k];

}

}

}

*if* ((i % 2 != 0) && (i == 3))

{

*for* (int j = matrSize / 2; j < matrSize; j++)

{

*for* (int k = matrSize / 2; k < matrSize; k++)

{

globalBlocks[i + 1][(j - (matrSize / 2)) \* (matrSize / 2) + (k - (matrSize / 2))] = A[j][k];

}

}

}

}

*for* (int i = 0; i < 25; ++i)

{

globalBlocks[0][i] = globalBlocks[1][i];

}

cout << "Result blocks: " << endl;

*for* (int l = 0; l < blockSize; l++)

{

cout << "Block #" << (l + 1) << ": ";

*for* (int m = 0; m < 25; m++)

{

*if* (m % 5 == 0)

{

cout << endl;

}

cout << globalBlocks[l][m] << " ";

}

cout << endl;

}

globalMax = globalBlocks[0][0];

globalMin = globalBlocks[0][0];

localMax = globalBlocks[0][0];

localMin = globalBlocks[0][0];

freeArray(A, matrSize);

}

MPI\_Barrier(MPI\_COMM\_WORLD);

*if* (min\_comm != MPI\_COMM\_NULL)

{

cout << "Counting local minimumes..." << endl;

MPI\_Comm\_rank(min\_comm, &subrank);

MPI\_Comm\_size(min\_comm, &subsize);

MPI\_Scatter(globalBlocks, 25, MPI\_DOUBLE, localVector, 25, MPI\_DOUBLE, ROOT, min\_comm);

localMin = getMin(25, localVector);

cout << "Rank #" << subrank << " local min: " << localMin << endl;

MPI\_Reduce(&localMin, &globalMin, 1, MPI\_DOUBLE, MPI\_MIN, ROOT, min\_comm);

}

*if* (max\_comm != MPI\_COMM\_NULL)

{

cout << "Counting local maximumes..." << endl;

MPI\_Comm\_rank(max\_comm, &subrank);

MPI\_Comm\_size(max\_comm, &subsize);

MPI\_Scatter(globalBlocks, 25, MPI\_DOUBLE, localVector, 25, MPI\_DOUBLE, ROOT, max\_comm);

localMax = getMax(25, localVector);

cout << "Rank #" << subrank << " local max: " << localMax << endl;

MPI\_Reduce(&localMax, &globalMax, 1, MPI\_DOUBLE, MPI\_MAX, ROOT, max\_comm);

}

*if* (rank == ROOT)

{

cout << endl

<< "Global min: " << globalMin << endl;

cout << endl

<< "Global max: " << globalMax << endl;

}

MPI\_Finalize();

*return* 0;

}

double \*\*allocate2DArray(int size)

{

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

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

{

a[i] = new double[size];

}

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

{

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

{

a[i][j] = 0;

}

}  
*return* a;

}

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

{

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

{

delete[] matrix[i];

}

delete[] matrix;

}

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

{

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

{

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

{

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

}

cout << endl;

}

cout << endl;

}

double getMin(int size, double \*vector)

{

double min = vector[0];

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

{

*if* (vector[i] < min)

{

min = vector[i];

}

}

*return* min;

}

double getMax(int size, double \*vector)

{

double max = vector[0];

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

{

*if* (vector[i] > max)

{

max = vector[i];

}

}

*return* max;

}

**Результаты выполнения программы**

На рисунке 1 представлен результат выполнения программы

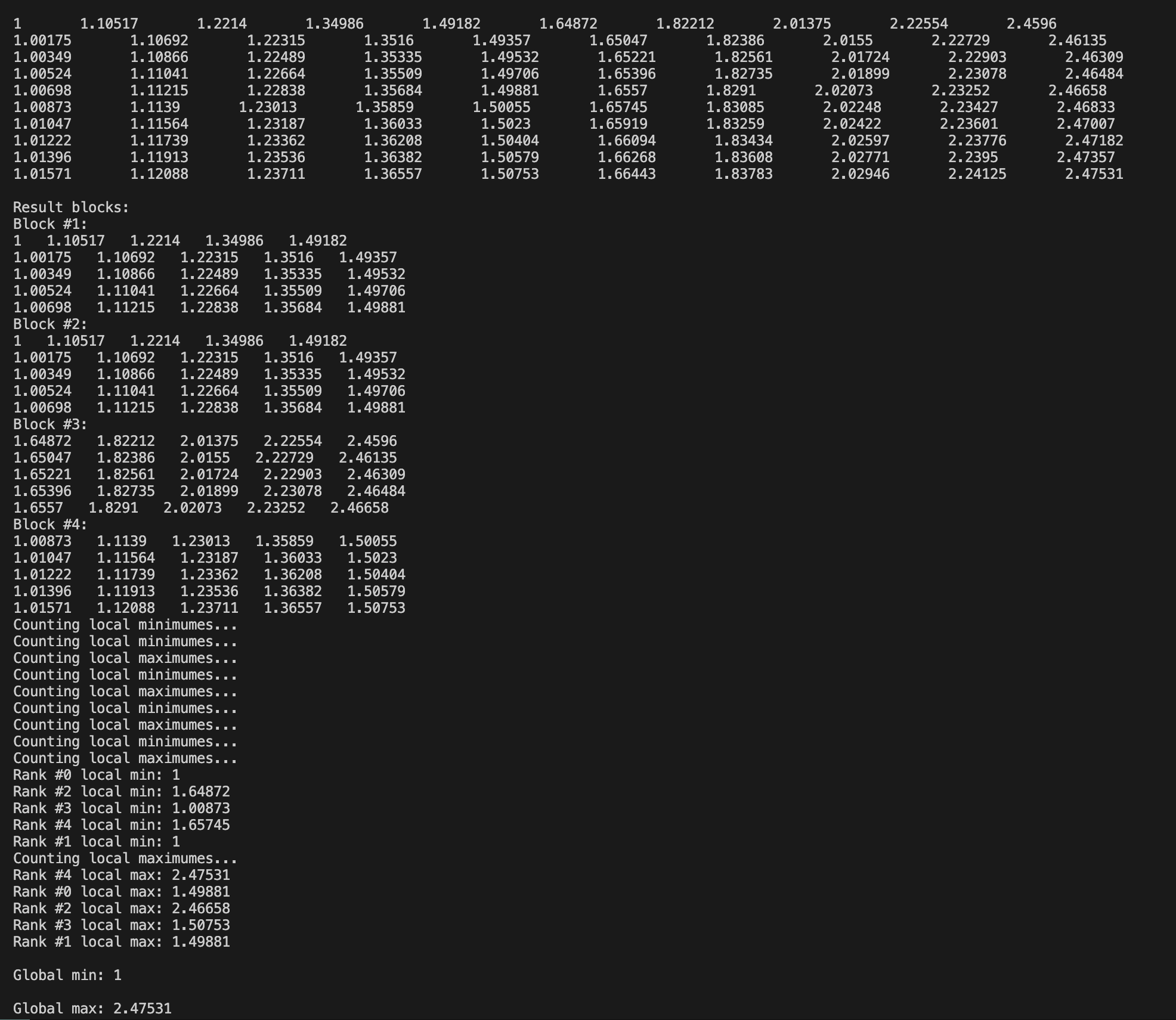


Рисунок 1 - Результат выполнения программы

**Выводы**

В ходе выполнения лабораторной работы были исследованы способы обмена данными между процессами в режиме широковещания или группового обмена с использованием MPI-функций.