**ЛАБОРАТОРНАЯ РАБОТА №3**

**ИСЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ ФОРМИРОВАНИЯ ВИР-**

**ТУАЛЬНЫХ ТОПОЛОГИЙ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ**

**Цель работы**

Исследовать возможности, предоставляемые MPI по формированию виртуальных топологий.

**Постановка задачи**

**Вариант №2**

Необходимо реализовать алгоритм перемножения матриц ленточным способом с распределением строк.

**Ход работы**

Была написана программа, решающая поставленную задачу. Листинг программы представлен ниже.

Листинг 1 – lab3.cpp

*#include* <iostream>

*#include* <iomanip>

*#include* <mpi.h>

using namespace std;

const int N = 4;

const int NUM\_DIMS = 1;

void showRow(int row[N]);

void showMatrix(int matrix[][N], string title);

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

{

int rank, processes;

MPI\_Init(&argc, &argv);

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

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

MPI\_Comm comm\_cart;

MPI\_Status status;

int masterProcess = 0;

int dims[NUM\_DIMS];

int periods[NUM\_DIMS], source, dest;

int reorder = 0;

int A[N][N];

int B[N][N];

int C[N][N];

int rowA[N], rowB[N], rowC[N];

int columnIndex = rank;

int M[4] = {0, 0, 0, 0};

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

{

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

{

C[i][j] = 0;

}

}

*// Обнуляем массив dims и заполняем массив periods для топологии "кольцо"*

*for* (int i = 0; i < NUM\_DIMS; i++)

{

dims[i] = 0;

periods[i] = 1;

}

*// Заполняем массив dims, где указываются размеры (одномерной) решетки*

MPI\_Dims\_create(processes, NUM\_DIMS, dims);

*// Создаем топологию "кольцо" с communicator(ом) comm\_cart*

MPI\_Cart\_create(MPI\_COMM\_WORLD, NUM\_DIMS, dims, periods, reorder, &comm\_cart);

*// Каждая ветвь находит своих соседей вдоль кольца, в направлении больших значений рангов*

MPI\_Cart\_shift(comm\_cart, 0, 1, &source, &dest);

*if* (rank == masterProcess)

{

int dataA[N][N] = {

{23, 93, 67, 35},

{90, 20, 55, 68},

{14, 96, 39, 66},

{44, 26, 88, 25},

};

int dataB[N][N] = {

{58, 66, 32, 12},

{44, 79, 16, 65},

{98, 31, 90, 73},

{27, 32, 66, 39},

};

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

{

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

{

A[i][j] = dataA[i][j];

}

}

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

{

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

{

B[i][j] = dataB[j][i];

}

}

showMatrix(A, "First:");

showMatrix(B, "Second:");

}

*// Разрезаем матрицы на строки и столбцы и передаем во все процессы для обработки*

MPI\_Scatter(A, N, MPI\_INT, rowA, N, MPI\_INT, masterProcess, MPI\_COMM\_WORLD);

*// MPI\_Scatter(B, 4, MPI\_INT, rowB, 4, MPI\_INT, masterProcess, MPI\_COMM\_WORLD);*

MPI\_Bcast(B, N \* N, MPI\_INT, masterProcess, MPI\_COMM\_WORLD);

*// Производим вычисления*

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

{

rowC[columnIndex] = 0;

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

{

rowC[columnIndex] += rowA[j] \* B[columnIndex][j];

}

cout << "Process " << rank << " : ";

showRow(rowA);

cout << "-- ";

showRow(B[columnIndex]);

cout << "-- " << rowC[columnIndex] << " -- " << rank << ":" << columnIndex << endl;

*if* (rank == masterProcess)

{

C[rank][columnIndex] = rowC[columnIndex];

*// showMatrix(C, "NOW looks like:");*

MPI\_Send(C, N \* N, MPI\_INT, (rank + 1), 0, MPI\_COMM\_WORLD);

MPI\_Recv(C, N \* N, MPI\_INT, processes - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

*else* *if* (rank == processes - 1)

{

MPI\_Recv(C, N \* N, MPI\_INT, rank - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

C[rank][columnIndex] = rowC[columnIndex];

*// showMatrix(C, "NOW looks like:");*

MPI\_Send(C, N \* N, MPI\_INT, masterProcess, 0, MPI\_COMM\_WORLD);

}

*else*

{

MPI\_Recv(C, N \* N, MPI\_INT, rank - 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

C[rank][columnIndex] = rowC[columnIndex];

*// showMatrix(C, "NOW looks like:");*

MPI\_Send(C, N \* N, MPI\_INT, (rank + 1), 0, MPI\_COMM\_WORLD);

}

*// Отправляем по кольцу следующий индекс столбца B*

MPI\_Send(&columnIndex, 1, MPI\_INT, dest, 12, comm\_cart);

MPI\_Recv(&columnIndex, 1, MPI\_INT, source, 12, comm\_cart, &status);

}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Barrier(comm\_cart);

*// MPI\_Gather(rowC, N, MPI\_INT, C, N, MPI\_INT, masterProcess, MPI\_COMM\_WORLD);*

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Barrier(comm\_cart);

*if* (rank == masterProcess)

{

showMatrix(C, "Result:");

}

*// Все ветви завершают системные процессы, связанные с топологией comm\_cart и завершают выполнение программы*

MPI\_Comm\_free(&comm\_cart);

MPI\_Finalize();

*return* 0;

}

void showRow(int row[N])

{

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

{

cout << row[i] << " ";

}

}

void showMatrix(int matrix[][N], string title)

{

cout << endl

<< title << endl;

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

{

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

{

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

}

cout << endl;

}

cout << endl;

}

**Результаты выполнения программы**

На рисунке 1 представлен результат выполнения программы.

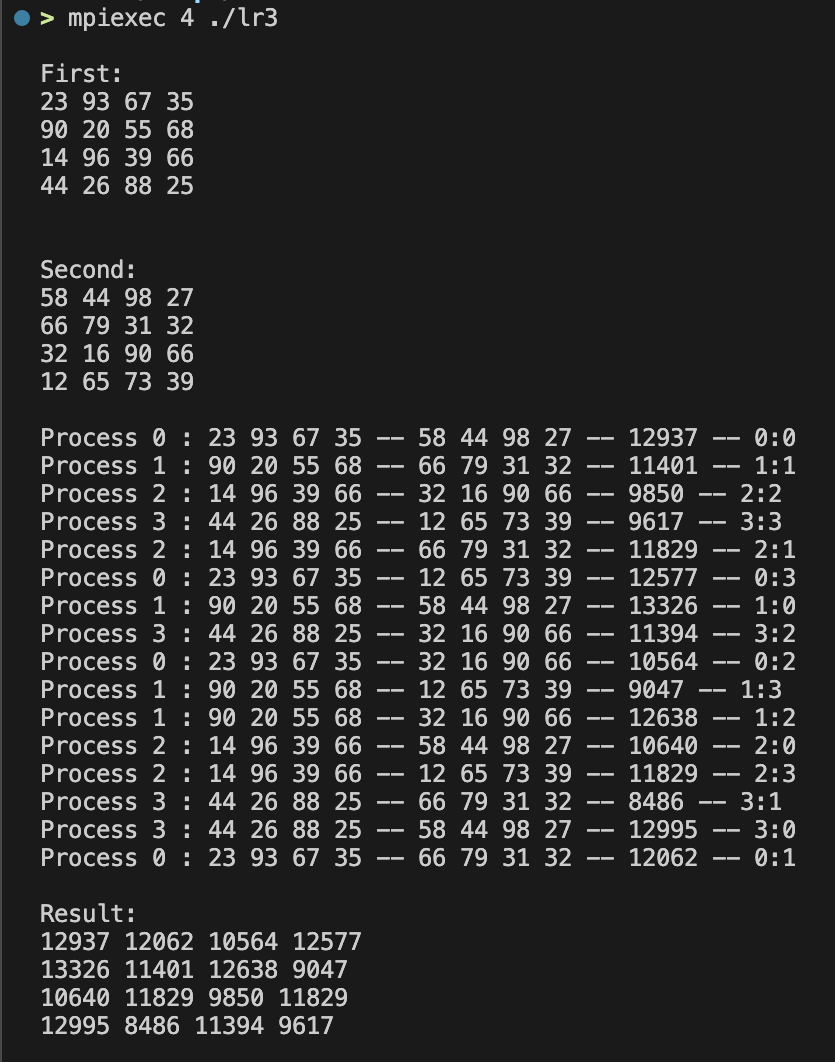


Рисунок 1 - Результат выполнения программы

**Выводы**

В ходе лабораторной работы были исследованы возможности, предоставляемые MPI по формированию виртуальных топологий.