Лабораторная работа №4

Производные типы данных в MPI

Максим Ромашкин

ПИН-34   Номер в списке: 18

Оглавление

[**Задание, код программы и пояснение:** 1](#_Toc95838769)

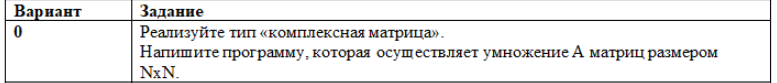
[**Листинг программы:** 1](#_Toc95838770)

[**Результат программы:** 4](#_Toc95838771)

[**Выполнение примера** 4](#_Toc95838772)

[**Ответы на контрольные вопросы** 4](#_Toc95838773)

# **Задание, код программ:**



# **Листинг программы:**

#include "mpi.h"

#include <iostream>

#include <queue>

using std::cout;

using std::endl;

using std::queue;

using std::min;

#define sizeMatrix 3

#define countMatrix 3

struct Complex {

int real;

int imaginary;

};

Complex\*\* initializationMatrix(int n) {

Complex\* data = new Complex[n \* n];

Complex\*\* array = new Complex \* [n];

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

array[i] = &(data[n \* i]);

}

return array;

}

void fillZeroMatrix(Complex\*\* array, int n) {

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

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

array[i][j].real = 0;

array[i][j].imaginary = 0;

}

}

}

void fillMatrix(Complex\*\* array, int n) {

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

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

array[i][j].real = i + j;

array[i][j].imaginary = i + j;

}

}

}

void printMatrix(Complex\*\* array, int n) {

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

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

cout << "(" << array[i][j].real << ", " << array[i][j].imaginary << "i" << ") ";

}

cout << endl;

}

}

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

int procNum, procRank;

MPI\_Init(&argc, &argv);

MPI\_Status status;

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

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

//регистрируем тип

MPI\_Datatype complex\_number;

MPI\_Datatype type[2] = { MPI\_INT, MPI\_INT };

int blocklen[2] = { 1, 1 };

MPI\_Aint disp[2] = { 0, 1 };

MPI\_Type\_create\_struct(2, blocklen, disp, type, &complex\_number);

MPI\_Type\_commit(&complex\_number);

queue<Complex\*\*> queueMatrix;

bool isEnd;

//Нулевой процесс

if (procRank == 0) {

//Инициализируем все матрицы и заполняем

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

Complex\*\* matrix;

matrix = initializationMatrix(sizeMatrix);

fillMatrix(matrix, sizeMatrix);

queueMatrix.push(matrix);

}

//Пока не отсанется результируещий матрицы отправляем для перемножения

while (queueMatrix.size() != 1) {

isEnd = false;

int size = (int)queueMatrix.size();

for (int i = 0; i < min((int)(size / 2), procNum - 1); i++) {

MPI\_Send(&isEnd, 1, MPI\_INT, i + 1, 0, MPI\_COMM\_WORLD);

Complex\*\* matrix1;

matrix1 = initializationMatrix(sizeMatrix);

matrix1 = queueMatrix.front();

queueMatrix.pop();

Complex\*\* matrix2;

matrix2 = initializationMatrix(sizeMatrix);

matrix2 = queueMatrix.front();

queueMatrix.pop();

MPI\_Send(&(matrix1[0][0]), 9, complex\_number, i + 1, 0, MPI\_COMM\_WORLD);

MPI\_Send(&(matrix2[0][0]), 9, complex\_number, i + 1, 0, MPI\_COMM\_WORLD);

}

queue<Complex\*\*> tmpQueueMatrix;

for (int i = 1; i <= min((int)(size / 2), procNum - 1); i++) {

Complex\*\* multiplyMatixResult;

multiplyMatixResult = initializationMatrix(sizeMatrix);

//Получаем результат перемножения

MPI\_Recv(&(multiplyMatixResult[0][0]), sizeMatrix \* sizeMatrix, complex\_number, i, 1, MPI\_COMM\_WORLD, &status);

tmpQueueMatrix.push(multiplyMatixResult);

}

while (queueMatrix.size()) {

tmpQueueMatrix.push(queueMatrix.front());

queueMatrix.pop();

}

queueMatrix = tmpQueueMatrix;

}

for (int i = 1; i < procNum; i++) {

isEnd = true;

MPI\_Send(&isEnd, 1, MPI\_INT, i, 0, MPI\_COMM\_WORLD);

}

//Выводим результат

printMatrix(queueMatrix.front(), sizeMatrix);

}

//Все процессы кроме 0

else {

while (true) {

MPI\_Recv(&isEnd, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, &status);

if (isEnd) {

break;

}

Complex\*\* matrix1;

Complex\*\* matrix2;

Complex\*\* multiplyMatixResult;

matrix1 = initializationMatrix(sizeMatrix);

matrix2 = initializationMatrix(sizeMatrix);

multiplyMatixResult = initializationMatrix(sizeMatrix);

MPI\_Recv(&(matrix1[0][0]), sizeMatrix \* sizeMatrix, complex\_number, 0, 0, MPI\_COMM\_WORLD, &status);

MPI\_Recv(&(matrix2[0][0]), sizeMatrix \* sizeMatrix, complex\_number, 0, 0, MPI\_COMM\_WORLD, &status);

fillZeroMatrix(multiplyMatixResult, sizeMatrix);

//Перемножаем матрицы

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

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

for (int k = 0; k < sizeMatrix; k++)

{

multiplyMatixResult[i][j].real += matrix1[i][k].real \* matrix2[k][j].real;

multiplyMatixResult[i][j].imaginary += matrix1[i][k].imaginary \* matrix2[k][j].imaginary;

}

}

}

//Отправляем результат перемножения 0 процессу

MPI\_Send(&(multiplyMatixResult[0][0]), sizeMatrix \* sizeMatrix, complex\_number, 0, 1, MPI\_COMM\_WORLD);

}

}

//удалаем свой тип

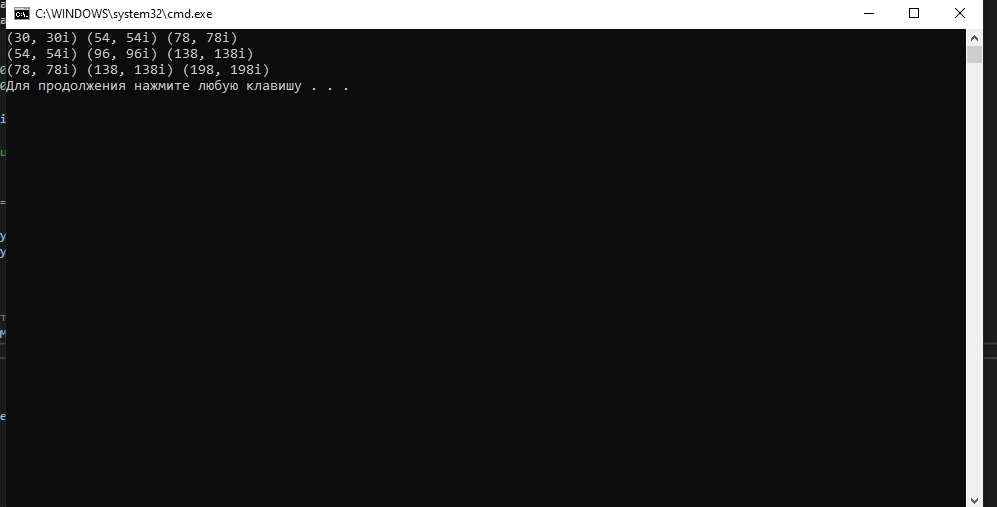
MPI\_Type\_free(&complex\_number);

MPI\_Finalize();

return 0;

}

# **Результат программы:**



# **Выполнение примера**

В разработке не было полноценных примеров

# **Ответы на контрольные вопросы**

В разработке не было контрольных вопросов