Ковальский Евгений

Лабораторная работа №4

Задание - Вариант 0.

**Задание.** Реализуйте тип «комплексная матрица». Напишите программу, которая осуществляет умножение A матриц размером NxN.

Код программы:

*//--------------------Подключаемые библиотеки--------------------//*

*/\*mpi\*/*

#include <headers/mpi.h>

*/\*std\*/*

#include <iostream>

#include <chrono>

#include <queue>

#include <deque>

#include <complex>

#include <vector>

*using* *namespace* std;

*//--------------------Структура* *для* *перемножения* *комплексных* *матриц--------------------//*

*struct* **ComplexMatrix**{

*//двумерный* *массив* *комлексных* *чисел*

vector<vector<complex<int>>> matrix;

*//количество* *элементов* *в* *ряду*

int rowSize;

*//конструктор* *создаёт* *матрицу* *rowSize* *\** *rowSize* *и* *заполняет* *её* *нулевыми* *комплексными* *значениями*

**ComplexMatrix**(int rowSize){

*this*->rowSize = rowSize;

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

vector<complex<int>> v;

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

v.push\_back(complex<int>(0, 0));

}

matrix.push\_back(v);

}

}

*/\** *конструктор* *создаёт* *матрицу* *rowSize* *\** *rowSize* *и* *заполняет* *её* *из* *массива* *сырых* *чисел*

*\** *массив* *сырых* *чисел* *содержит* *пары* *элементов* *r,* *im,* *r,* *im* *...* *\*/*

**ComplexMatrix**(int\* m, int rowSize){

*this*->rowSize = rowSize;

int k = 0, l = 1;

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

vector<complex<int>> v;

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

v.push\_back(complex<int>(m[k], m[l]));

k += 2;

l += 2;

}

matrix.push\_back(v);

}

}

*//умножение* *двух* *матриц*

ComplexMatrix *operator*\* (ComplexMatrix rhs){

int rhvRowSize = rhs.rowSize;

ComplexMatrix res(rhvRowSize);

*//самый* *стандартный* *алгоритм* *умножения* *матриц*

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

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

res.matrix[i][j] = 0;

*for* (int k = 0; k < rhvRowSize; ++k) {

res.matrix[i][j] += matrix[i][k] \* rhs.matrix[k][j];

*//* *cout* *<<* *matrix[i][k].real()* *<<* *"+"* *<<* *matrix[i][k].imag()* *<<* *"i* *\** *"*

*//* *<<* *rhs.matrix[k][j].real()* *<<* *"+"* *<<* *rhs.matrix[k][j].imag()* *<<* *"i* *=* *"*

*//* *<<* *res.matrix[i][j].real()* *<<* *"+"* *<<* *res.matrix[i][j].imag()* *<<* *endl;*

}

}

}

*return* res;

}

*//переводит* *матрицу* *в* *массив* *сырых* *чисел,* *предварительно* *инициализированный* *пользователем*

*//числа* *идут* *в* *последовательности* *r,* *im,* *r,* *im* *...*

void **getRaw**(int\* m){

int k = 0, l = 1;

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

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

m[k] = matrix[i][j].real();

m[l] = matrix[i][j].imag();

k += 2;

l += 2;

}

}

}

};

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

{

*//--------------------Инициализация* *MPI--------------------//*

int procNum, procRank;

MPI\_Status status;

MPI\_Init(&argc, *&argv*);

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

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

*//--------------------Инициализация* *матриц--------------------//*

*const* int matrixSize = 3;

*const* int matrixAmount = 4;

*const* int matrixTypeElemNum = matrixSize \* matrixSize \* 2;

int matrices[matrixAmount][matrixTypeElemNum];

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

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

matrices[i][j] = (i + j) % 5;

}

}

*//--------------------Проверка* *вычислений* *без* *параллельного* *режима--------------------//*

*if*(procRank == 0){

ComplexMatrix matrixRes(matrixSize);

ComplexMatrix matrix1(*matrices[*0*]*, matrixSize);

ComplexMatrix matrix2(*matrices[*1*]*, matrixSize);

matrixRes = matrix2 \* matrix1;

*for*(int i = 2; i < matrixAmount; ++i){

ComplexMatrix matrix3(*matrices[i]*, matrixSize);

matrixRes = matrix3 \* matrixRes;

}

*//выводим* *итоговую* *матрицу*

int tinyCounter = 0;

int\* rawMatrix = *new* int[matrixTypeElemNum];

matrixRes.getRaw(*rawMatrix*);

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

cout << rawMatrix[i] << " ";

++tinyCounter;

*if*(tinyCounter == matrixSize \* 2){

cout << endl;

tinyCounter = 0;

}

}

}

*auto* start = chrono::steady\_clock::now();

*//--------------------Инициализация* *MPI* *типа* *матрицы--------------------//*

MPI\_Datatype matrixType;

MPI\_Type\_contiguous(matrixTypeElemNum, MPI\_INT, *&matrixType*);

MPI\_Type\_commit(*&matrixType*);

*//--------------------Инициализация* *матриц* *для* *вычислений--------------------//*

int recvMatrix[matrixTypeElemNum];

int m1[matrixTypeElemNum];

int m2[matrixTypeElemNum];

bool endFlag = *true*;

*//--------------------Главный* *процесс--------------------//*

*if* (procRank == 0)

{

*//копируем* *матрицы*

int matricesToCalc[matrixAmount][matrixTypeElemNum];

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

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

matricesToCalc[i][j] = matrices[i][j];

}

}

*//значение,* *показывающее* *количество* *оставшихся* *для* *перемножения* *матриц*

int totalMatricesLeft = matrixAmount;

*/\** *значение,* *показывающее* *текущее* *количество* *оставшихся* *для* *перемножения* *матриц*

*\** *с* *учётом* *процессов,* *выполняемых* *в* *данный* *момент* *\*/*

int matricesLeft = matrixAmount;

*//значение* *для* *коммуникации* *между* *процессами*

int sendProcNum;

*//очередь* *из* *свободных* *процессов*

queue <int> freeProcs;

*//очередь* *из* *занятых* *процессов*

deque <int> busyProcs;

*//изначально* *все* *процессы* *свободны*

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

freeProcs.push(i);

}

*//цикл,* *определяющий,* *все* *ли* *матрицы* *перемножены*

*while* (totalMatricesLeft != 1)

{

*/\** *цикл,* *выдающий* *задания* *другим* *процессам,* *пока* *среди* *них* *есть* *свободные*

*\** *или* *не* *закончатся* *матрицы* *для* *перемножения* *\*/*

*while* (matricesLeft > 1 && freeProcs.size() != 0)

{

*//берём* *первый* *из* *свободных* *процессов* *в* *очереди*

sendProcNum = freeProcs.front();

freeProcs.pop();

*//помечаем,* *что* *процесс* *занят* *для* *того,* *чтобы* *запомнить* *последовательность* *умножения* *матриц*

busyProcs.push\_back(sendProcNum);

cout << "main processor send messages to " << sendProcNum << endl;

*//отправляем* *информацию* *том,* *завершены* *ли* *вычисления* *выполняемому* *процессу*

MPI\_Send(&endFlag, 1, MPI\_C\_BOOL, sendProcNum, 0, MPI\_COMM\_WORLD);

*//отправляем* *две* *последние* *матрицы* *из* *списка* *матриц* *для* *перемножения* *выполнямому* *процессу*

MPI\_Send(matricesToCalc[matricesLeft-1], 1, matrixType, sendProcNum, 0, MPI\_COMM\_WORLD);

MPI\_Send(matricesToCalc[matricesLeft-2], 1, matrixType, sendProcNum, 0, MPI\_COMM\_WORLD);

*//матриц* *для* *перемножения* *становится* *на* *2* *меньше*

matricesLeft -= 2;

*//всего* *остаётся* *на* *1* *матрицу* *меньше* *после* *умножения,* *A\*B* *становятся* *матрицей* *C*

--totalMatricesLeft;

}

*while*(busyProcs.size() != 0){

*//берём* *номер* *последнего* *занятого* *процесса,* *чтобы* *получить* *перемноженные* *матрицы* *в* *обратном* *порядке*

sendProcNum = busyProcs.back();

busyProcs.pop\_back();

cout << "main processor waiting for messages from other processors" << endl;

*//ожидаем* *сообщение* *от* *выполняющих* *процессов,* *содержащее* *вычисленную* *матрицу*

MPI\_Recv(*recvMatrix*, 1, matrixType, sendProcNum, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &*status*);

cout << "main processor took message from processor " << sendProcNum << endl;

*//добавляем* *процесс* *в* *очередь* *свободных*

freeProcs.push(sendProcNum);

*//добавляем* *матрицу* *в* *очередь* *на* *умножение*

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

matricesToCalc[matricesLeft][i] = recvMatrix[i];

}

*//теперь* *для* *вычисления* *есть* *на* *одну* *матрицу* *больше*

++matricesLeft;

}

}

*//после* *завершения* *цикла* *говорим* *всем* *процессам* *прекращать* *работу*

endFlag = *false*;

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

MPI\_Send(&endFlag, 1, MPI\_C\_BOOL, i, 0, MPI\_COMM\_WORLD);

}

*//выводим* *итоговую* *матрицу*

int tinyCounter = 0;

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

cout << matricesToCalc[0][i] << " ";

++tinyCounter;

*if*(tinyCounter == matrixSize \* 2){

cout << endl;

tinyCounter = 0;

}

}

}

*//--------------------Остальные* *процессы--------------------//*

*else* {

*//цикл,* *прекращающий* *работу* *когда* *endFlag* *становится* *false*

*while* (*true*) {

cout << "processor " << procRank << " wating message from main processor" << endl;

*//получаем* *информацию* *о* *том,* *завершены* *ли* *вычисления*

MPI\_Recv(*&endFlag*, 1, MPI\_C\_BOOL, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, *&status*);

*//если* *не* *завершены*

*if* (endFlag) {

*//получаем* *две* *матрицы* *для* *перемножения*

MPI\_Recv(*m1*, 1, matrixType, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, *&status*);

MPI\_Recv(*m2*, 1, matrixType, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, *&status*);

*//перемножаем* *матрицы*

ComplexMatrix m1\_c(*m1*, matrixSize);

ComplexMatrix m2\_c(*m2*, matrixSize);

ComplexMatrix recvMatrix\_c(matrixSize);

recvMatrix\_c = m1\_c \* m2\_c;

recvMatrix\_c.getRaw(*recvMatrix*);

cout << "processor " << procRank << " done calculations and send result" << endl;

*//отправляем* *готовую* *матрицу* *и* *номер* *выполняющего* *процесса*

MPI\_Send(recvMatrix, 1, matrixType, 0, 0, MPI\_COMM\_WORLD);

}

*//если* *вычисления* *завершены,* *завершаем* *работу*

*else* {

cout << "processor " << procRank << " end" << endl;

*break*;

}

}

}

*//--------------------Завершение* *работы* *MPI--------------------//*

MPI\_Type\_free(*&matrixType*);

MPI\_Finalize();

*//выводим* *время* *работы*

*auto* end = chrono::steady\_clock::now();

chrono::duration<double> elapsed\_seconds = end-start;

cout << "elapsed time: " << elapsed\_seconds.count()/1000 << "s\n";

*return* 0;

}

Вывод программы (4 процесса):

-1118 -839 -2749 -423 -1570 48

-1405 -606 -3026 250 -1607 451

-1077 -363 -2173 328 -1149 454

processor 1 wating message from main processor

processor 3 wating message from main processor

processor 2 wating message from main processor

main processor send messages to 1

main processor send messages to 2

main processor waiting for messages from other processors

processor 1 done calculations and send result

processor 2 done calculations and send result

processor 1 wating message from main processor

processor 2 wating message from main processor

main processor took message from processor 2

main processor waiting for messages from other processors

main processor took message from processor 1

main processor send messages to 3

main processor waiting for messages from other processors

processor 3 done calculations and send result

processor 3 wating message from main processor

main processor took message from processor 3

processor 1 end

processor 2 end

processor 3 end

-1118 -839 -2749 -423 -1570 48

-1405 -606 -3026 250 -1607 451

-1077 -363 -2173 328 -1149 454

elapsed time: 3.4162e-06s

elapsed time: 3.3698e-06s

elapsed time: 3.1788e-06s

elapsed time: 3.4704e-06s