Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Самарский национальный исследовательский университет

имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

**Отчет по курсовой работе**

Дисциплина: «Параллельные алгоритмы»

Вариант № 5

Выполнили студенты: Курицын Н.С.

Группа 6132-010304D

Преподаватель Головашкин Д.Л.

2022

**Исходные данные**

**Условие задачи:**

Разработать программу численного решения задачи теплопроводности в тонком однородном кольце радиусом R, сечением s на временном промежутке , если на поверхности кольца происходит теплообмен с окружающей средой, описываемый законом Ньютона с коэффициентом теплообмена *α*. Коэффициенты теплопроводности и объемной теплоемкости материала кольца равны *k* , и *c* соответственно.

Начальное распределение температуры кольца описывает функцией

где – полярный угол.

Предполагается, что в кольце отсутствует внутренние источники тепла.

Для численного решения краевой задачи использовать:

* модифицированную явную конечно-разностную схему (Курицын Н.С).

При проведении расчетов использовать значения параметров , а также выражение функции , указанные преподавателем.

Значения параметров, указанные преподавателем:

* ;
* ;
* ;
* ;

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc153205216)

[1 Математическая постановка задачи 5](#_Toc153205217)

[2 Модифицированная явная конечно-разностная схема (Курицын Н.С) 6](#_Toc153205218)

[2.1 Построение явной конечно-разностной схемы 6](#_Toc153205219)

[3 Результаты вычислительных экспериментов на последовательной программе 7](#_Toc153205220)

[3.1 Графическое подтверждение сходимости разностного уравнения к точечному 7](#_Toc153205221)

[4 Реализация параллельной программы с OPenmp 8](#_Toc153205222)

[5 Реализация параллельной программы с MPI 10](#_Toc153205223)

[ЗАКЛЮЧЕНИЕ 12](#_Toc153205224)

# **ВВЕДЕНИЕ**

Характеризуя метод конечных разностей необходимо выделить его достоинства и недостатки в сравнении с другими методами.

К достоинствам метода конечных разностей следует отнести его высокую универсальность, например, значительно более высокую, чем у аналитических методов. Применение этого метода нередко характеризуется относительной простотой построения решающего алгоритма и его программной реализации. Зачастую удается осуществить распараллеливание решающего алгоритма.

К числу недостатков метода следует отнести: проблематичность его использования на нерегулярных сетках; очень быстрый рост вычислительной трудоемкости при увеличении размерности задачи (увеличении числа неизвестных переменных); сложность аналитического исследования свойств разностной схемы.

Суть метода конечных разностей состоит в замене исходной (непрерывной) задачи математической физики ее дискретным аналогом (разностной схемой), а также последующим применением специальных алгоритмов решения дискретной задачи.

В настоящей работе метод конечных разностей применен для численного решения задачи теплопроводности. Вычислительный алгоритм основан на использовании явной разностной схемы. Проведено теоретическое исследование аппроксимации и устойчивости разностных схем. Сделан вывод о сходимости сеточного решения к точному решению исходной задачи. Разработана компьютерная программа. Приведены графические результаты численного решения задачи теплопроводности.

1. Математическая постановка задачи

В общем виде уравнение теплопроводности выглядит следующим образом:

. (1.1)

Оно будет действовать на всей длине кольца. Тогда уравнение действует при .

Для удобства представим кольцо в виде трубки.

Так как мы разделили кольцо, то левый и правый концы трубки будут одинаково распределять тепло:

. (1.2)

Из-за разделения кольца изменение распределения тепла, также одинаково на двух концах:

. (1.3)

В условии дан закон, по которому можно определить концентрацию вещества в трубке в начальный момент времени:

. (1.4)

Таким образом, имеем следующую систему:

(1.5)

1. Модифицированная явная конечно-разностная схема (Курицын Н.С)

# **2.1 Построение явной конечно-разностной схемы**

Для построения простейшей явной конечно-разностной схемы задачи (1.5) зададим равномерную сетку, которая выглядит следующим образом:

(2.1)

Где – шаг разбиения по *x*, а – шаг разбиения по *t*.

Запишем приближенные выражения для производных, входящих в задачу (1.5):

, (2.2)

. (2.3)

В результате получаем явную схему:

(2.4)

где – сеточный аналог функций , а – искомая сеточная функция, которую будем рассматривать как некоторое приближение к решению задачи (1.5).

1. Результаты вычислительных экспериментов на последовательной программе

**3.1 Графическое подтверждение сходимости разностного уравнения к точечному**

На рисунке 1 приведены полученные графики, подтверждающие сходимость разностного уравнения решения дискретной задачи (2.4) к аналитическому решению исходной задачи (1.5) на сгущающихся сетках.

Для этого необходимо зафиксировать значения времени и радиуса. Для данного эксперимента зафиксировано *t = 200, R = 7000*.

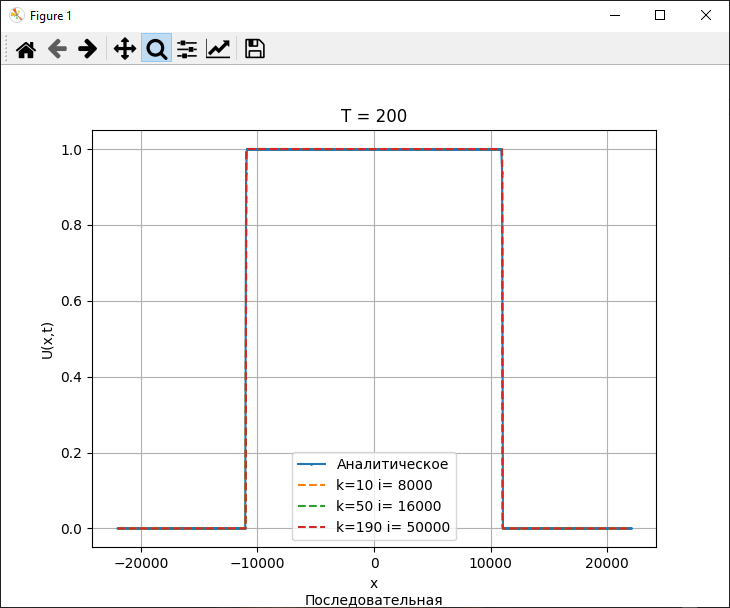


Рисунок 1 – График, демонстрирующий сходимость решения по времени *t*

1. Реализация параллельной программы с OPenmp

В ходе лабораторной работы также реализована программа на языке программирования C++ с использованием стандарта распараллеливания программ OpenMP.

Параллельный алгоритм решения явной схемы заключается в том, что 2 процесса начинают вычисление на одном уровне по времени, но только со своими частями по *x.* OpenMP поддерживает автоматическое распараллеливание, поэтому он сам будет обмениваться соседними значениями между задачами.

Для визуализации вычисленной схемы можно построить графики, запустив скрипт на python. Должны быть установлены библиотеки numpy и matplotlib.

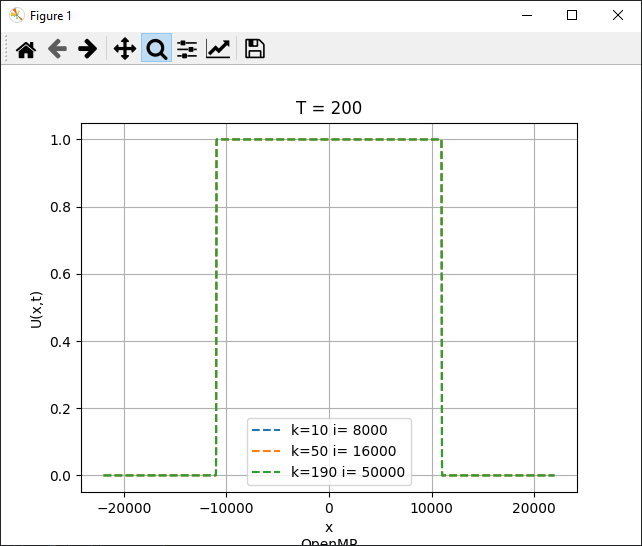


Рисунок 2 – График, демонстрирующий сходимость решения по времени *t* на OpenMP

Замеры времени работы программы делались по 12 запускам. Результаты работы OpenMP представлены в таблице 1.

Таблица 1 – Результаты ускорения параллельной программы с OpenMP относительно последовательно с использованием различного числа нитей.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время работы последовательной программы, с | Время работы параллельной программы, с | Ускорение |
| (8000,10) | 0.000166 | 0.0003 | 0,55 |
| (16000,50) | 0.0045 | 0.003 | 1,5 |
| (50000,190) | 0.05125 | 0.03125 | 1,64 |

Из таблицы видно, что при увеличении *I* увеличивается ускорение, так как каждая задача забирает равное количество элементов *I* и дальше бежит по всем уровням *K*, обмениваясь значениями на предыдущем слое. Чем меньше *K,* тем меньшекоммуникаций между задачами.

1. Реализация параллельной программы с MPI

В ходе лабораторной работы также реализована программа на языке программирования C++ с использованием стандарта распараллеливания программ MPI. Обмен с использованием технологии MPI становится проще, так как функция MPI\_Recv является блокирующей, то есть, мы вручную можем не синхронизовать потоки, MPI делает это за нас.

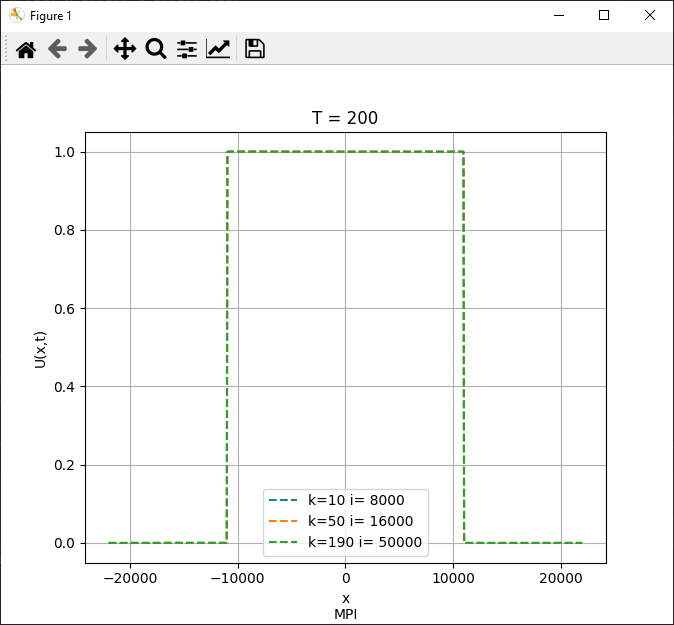


Рисунок 3 – График, демонстрирующий сходимость решения по времени *t* на MPI

Замеры времени работы программы делались по 12 запускам. Результаты MPI представлены в таблице 2.

Таблица 2 – Результаты ускорения параллельной программы с MPI относительно последовательно с использованием различного числа нитей.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Время работы последовательной программы, с | Время работы параллельной программы, с | Ускорение |
| (8000,10) | 0.000166 | 0,00012 | 1,383 |
| (16000,50) | 0.0045 | 0,00231 | 1,948 |
| (50000,190) | 0.05125 | 0.0233 | 2,199 |

Из таблицы видно, что MPI дает большее ускорение, чем OpenMP.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы осуществлена постановка краевой задачи для уравнения теплопроводности. Для численного решения краевой задачи построена явная модифицированная схема. Были написаны параллельные алгоритмы расчёта схемы на языке C++ с использованием технологий OpenMP и MPI, а также последовательная программа.

Результаты экспериментов, показывающие наилучшую производительность кода при использовании MPI, так при размерности сетки по удалось достичь ускорение 2,199, а на OpenMP всего 1,64. Это объясняется тем, что при использовании технологии MPI время затрачивается только на пересылку данных, а выполнение операций проходит параллельно и без задержек.

**ПРИЛОЖЕНИЕ А**

**Код последовательной программы**

#include <iostream>

#include <cmath>

#include "SFML/Graphics.hpp"

#include <fstream>

#include <string>

using namespace std;

const double PI = 3.141592653589793238463;

const double k = 0.59;

const double c = 1.65;

const double R = 7000;

const double T = 200;

const double const S = 0.04;

const double L = 2 \* PI \* R;

const int I\_array[] = { 8000, 16000, 50000 };

const int K\_array[] = { 10 , 50, 190 };

const double a = sqrt(k / c);

const string extension = ".txt";

double phi(double x) {

if (-PI \* R / 2 <= x && x <= PI \* R / 2) {

return 1.0;

}

return 0.0;

}

void writeArray(string filename, double\* my\_array,int size) {

ofstream outputFile(filename);

if (outputFile.is\_open()) {

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

outputFile << my\_array[i] << " ";

}

outputFile.close();

std::cout << "Массив успешно записан в файл " << filename << std::endl;

}

else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

}

void print\_Array(string name\_array, double\* my\_array, int size) {

cout << name\_array << "---->>> "<<endl;

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

cout << my\_array[i] << " ";

}

cout << endl;

}

int main() {

double step\_x;

double step\_t;

int length = sizeof(I\_array) / sizeof(I\_array[0]);

double\* massive\_time = new double[length];

for (int count = 0; count < length; count++) {

massive\_time[count] = 0.0;

}

for (int step = 0; step < 12; step++) {

for (int count = 0; count < length; count++) {

double step\_x = (2 \* PI \* R) / (I\_array[count] - 1);

double step\_t = T / (K\_array[count] - 1);

double\* x\_array = new double[I\_array[count]];

double\* t\_array = new double[K\_array[count]];

for (int i = 0; i < I\_array[count]; i++) {

x\_array[i] = -PI \* R + i \* step\_x;

}

for (int i = 0; i < K\_array[count]; i++) {

t\_array[i] = 0 + i \* step\_t;

}

int I = I\_array[count] - 1;

int K = K\_array[count] - 1;

double gamma = (k \* step\_t) / (c \* pow(step\_x, 2));

double\*\* u = new double\* [2];

for (int j = 0; j < 2; j++) {

u[j] = new double[I + 1];

}

for (int j = 0; j < 2; j++){

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

if (j == 0) {

u[j][i] = phi(x\_array[i]);

}

else {

u[j][i] = 0.0;

}

}

}

int nubmer\_new = 0;

int nubmer\_old = 0;

clock\_t start\_time\_parallel = clock();

for (int k = 0; k < K; k++) {

nubmer\_new = (k % 2 == 0) ? 1 : 0;

nubmer\_old = (k % 2 != 0) ? 1 : 0;

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

u[nubmer\_new][i] = u[nubmer\_old][i] + gamma \* (u[nubmer\_old][i + 1] - 2 \* u[nubmer\_old][i] + u[nubmer\_old][i - 1]);

}

u[nubmer\_new][I] = (u[nubmer\_old][1] + u[nubmer\_old][I - 1]) / 2;

u[nubmer\_new][0] = u[nubmer\_new][I];

}

clock\_t end\_time\_parallel = clock();

double duration = static\_cast<double>(end\_time\_parallel - start\_time\_parallel) / CLOCKS\_PER\_SEC;

cout << endl;

cout << I\_array[count] << " " << K\_array[count] <<" Time - >" << duration << endl;

massive\_time[count] += duration;

/\*string name\_file\_x\_array = "E:\\study\\Мага\\Параллельные\\arrays\\x\_array\_" + to\_string(count) + extension;

string name\_file\_u\_t\_array = "E:\\study\\Мага\\Параллельные\\arrays\\u\_t\_array\_" + to\_string(count) + extension;

string name\_file\_t\_array = "E:\\study\\Мага\\Параллельные\\arrays\\t\_array\_" + to\_string(count) + extension;

writeArray(name\_file\_x\_array, x\_array, I\_array[count]);

writeArray(name\_file\_u\_t\_array, u[nubmer\_new], I\_array[count]);

writeArray(name\_file\_t\_array, t\_array, K\_array[count]);\*/

delete[] x\_array;

delete[] t\_array;

for (int j = 0; j < 2; j++) {

delete[]u[j];

}

delete[]u;

}

cout << endl << endl;

}

for (int count = 0; count < length; count++) {

cout << I\_array[count] << " " << K\_array[count] << " " << massive\_time[count] / 12 << endl;

}

return 0;

}

**ПРИЛОЖЕНИЕ Б**

**Код OpenMP программы**

#include <iostream>

#include <omp.h>

#include <cmath>

#include <ctime>

#include <fstream>

#include <string>

using namespace std;

const double PI = 3.141592653589793238463;

const double k = 0.59;

const double c = 1.65;

const double R = 7000;

const double T = 200;

const double const S = 0.04;

const double L = 2 \* PI \* R;

const int I\_array[] = { 8000, 16000, 50000 };

const int K\_array[] = { 10 , 50, 190 };

const double a = sqrt(k / c);

const string extension = ".txt";

double phi(double x) {

if (-PI \* R / 2 < x && x < PI \* R / 2) {

return 1.0;

}

return 0.0;

}

void writeArray(string filename, double\* my\_array, int size) {

ofstream outputFile(filename);

if (outputFile.is\_open()) {

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

outputFile << my\_array[i] << " ";

}

outputFile.close();

std::cout << "Массив успешно записан в файл " << filename << std::endl;

}

else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

}

void print\_Array(string name\_array, double\* my\_array, int size) {

cout << name\_array << "---->>> " << endl;

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

cout << my\_array[i] << " ";

}

cout << endl;

}

string areArraysEqual(double\*\* u\_1, double\*\* u\_2, int K, int I) {

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

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

if (u\_1[i][j] != u\_2[i][j]) return "Не равны!!!";

};

}

return "Равны";

}

int main()

{

setlocale(LC\_ALL, "Russian");

int num\_threads = 2;

int length = sizeof(I\_array) / sizeof(I\_array[0]);

//omp\_set\_num\_threads(num\_threads);

cout << "Всего потоков - > " << num\_threads << endl;

double\* massive\_time = new double[length];

for (int step = 0; step < length; step++) {

massive\_time[step] = 0.0;

}

for (int step = 0; step < 12; step++) {

for (int count = 0; count < length; count++) {

double step\_x = (2 \* PI \* R) / (I\_array[count] - 1);

double step\_t = T / (K\_array[count] - 1);

double\* x\_array = new double[I\_array[count]];

double\* t\_array = new double[K\_array[count]];

for (int i = 0; i < I\_array[count]; i++) {

x\_array[i] = -PI \* R + i \* step\_x;

}

for (int i = 0; i < K\_array[count]; i++) {

t\_array[i] = 0 + i \* step\_t;

}

int I = I\_array[count] - 1;

int K = K\_array[count] - 1;

cout << I\_array[count] << " " << K\_array[count] << endl;;

omp\_set\_num\_threads(num\_threads);

/\*for (int k = 0; k < K; k++) {

#pragma omp parallel for shared(u\_new,u\_old)

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

u\_new[i] = u\_old[i] + gamma \* (u\_old[i + 1] - 2 \* u\_old[i] + u\_old[i - 1]);

}

u\_new[I] = (u\_old[1] + u\_old[I - 1]) / 2;

u\_new[0] = u\_new[I];

#pragma omp parallel for shared(u\_new,u\_old)

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

u\_old[i] = u\_new[i];

}

}\*/

double\*\* u = new double\* [2];

for (int j = 0; j < 2; j++) {

u[j] = new double[I + 1];

}

for (int j = 0; j < 2; j++) {

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

if (j == 0) {

u[j][i] = phi(x\_array[i]);

}

else {

u[j][i] = 0.0;

}

}

}

double gamma = (k \* step\_t) / (c \* pow(step\_x, 2));

int nubmer\_new = 0;

int nubmer\_old = 0;

clock\_t start\_time\_parallel = clock();

for (int k = 0; k < K; k++) {

nubmer\_new = (k % 2 == 0) ? 1 : 0;

nubmer\_old = (k % 2 != 0) ? 1 : 0;

//cout << " u\_new-> " << nubmer\_new << " u\_old-> " << nubmer\_old << endl;

#pragma omp parallel for shared(u)

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

u[nubmer\_new][i] = u[nubmer\_old][i] + gamma \* (u[nubmer\_old][i + 1] - 2 \* u[nubmer\_old][i] + u[nubmer\_old][i - 1]);

}

#pragma omp sections

{

#pragma omp section

u[nubmer\_new][I] = (u[nubmer\_old][1] + u[nubmer\_old][I - 1]) / 2;

#pragma omp section

u[nubmer\_new][0] = u[nubmer\_new][I];

}

}

clock\_t end\_time\_parallel = clock();

// Рассчитываем разницу между начальным и конечным временем

double duration\_parallel = static\_cast<double>(end\_time\_parallel - start\_time\_parallel) / CLOCKS\_PER\_SEC;

cout << " Time - >" << duration\_parallel << endl;

massive\_time[count] += duration\_parallel;

/\*string name\_file\_x\_array = "E:\\study\\Мага\\Параллельные\\arrays\\x\_array\_OpenMP\_" + to\_string(count) + extension;

string name\_file\_u\_t\_array = "E:\\study\\Мага\\Параллельные\\arrays\\u\_t\_array\_OpenMP\_" + to\_string(count) + extension;

string name\_file\_t\_array = "E:\\study\\Мага\\Параллельные\\arrays\\t\_array\_OpenMP\_" + to\_string(count) + extension;

writeArray(name\_file\_x\_array, x\_array, I\_array[count]);

writeArray(name\_file\_u\_t\_array, u[nubmer\_new], I\_array[count]);

writeArray(name\_file\_t\_array, t\_array, K\_array[count]);\*/

delete[] x\_array;

delete[] t\_array;

for (int j = 0; j < 2; j++) {

delete[]u[j];

}

delete[]u;

}

cout << endl << endl;

}

cout << "Среднее время:\n";

for (int step = 0; step < length; step++) {

cout<<I\_array[step]<<" "<< K\_array[step]<<" "<<massive\_time[step]/12<<endl;

}

return 0;

}

**ПРИЛОЖЕНИЕ В**

**Код MPI программы**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#include "iostream"

#include <ctime>

#include <fstream>

#include <string>

using namespace std;

const double PI = 3.141592653589793238463;

const double k = 0.59;

const double c = 1.65;

const double R = 7000;

const double T = 200;

const double const S = 0.04;

const double L = 2 \* PI \* R;

const int I\_array[] = { 50000 };

const int K\_array[] = { 190 };

const double a = sqrt(k / c);

const string extension = ".txt";

double phi(double x) {

if (-PI \* R / 2 <= x && x <= PI \* R / 2) {

return 1.0;

}

return 0.0;

}

void writeArray(string filename, double\* my\_array, int size) {

ofstream outputFile(filename);

if (outputFile.is\_open()) {

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

outputFile << my\_array[i] << " ";

}

outputFile.close();

std::cout << "Массив успешно записан в файл " << filename << std::endl;

}

else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

}

void writeArray\_2d(string filename, double\*\* my\_array, int K,int I) {

ofstream outputFile(filename);

if (outputFile.is\_open()) {

for (int k = 0; k <= K; k++) {

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

outputFile << my\_array[k][i] << " ";

}

outputFile<<"\n";

}

outputFile.close();

std::cout << "Массив успешно записан в файл " << filename << std::endl;

}

else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

}

void print\_Array(string name\_array, double\* my\_array, int size) {

cout << name\_array << "---->>> " << endl;

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

cout << my\_array[i] << " ";

}

cout << endl;

}

string areArraysEqual(double\*\* u\_1, double\*\* u\_2, int K, int I) {

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

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

if (u\_1[i][j] != u\_2[i][j]) return "Не равны!!!";

};

}

return "Равны";

}

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

int length = sizeof(I\_array) / sizeof(I\_array[0]);

for (int count = 0; count < length; count++) {

int I = I\_array[count] - 1;

int K = K\_array[count] - 1;

double step\_x = (2 \* PI \* R) / (I\_array[count] - 1);

double step\_t = T / (K\_array[count] - 1);

double gamma = (k \* step\_t) / (c \* pow(step\_x, 2));

MPI\_Init(&argc, &argv);

int rank, size;

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

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

cout << I << " " << K << endl;

int size\_for\_proc = (I + 1) / size;

int start\_in\_proc = rank \* size\_for\_proc;

int end\_in\_proc = start\_in\_proc + size\_for\_proc - 1;

//cout << "Proc " << rank << " start = " << start\_in\_proc << " end = " << end\_in\_proc << " size = " << size\_for\_proc << endl;

int rank\_per\_gran;

int rank\_per\_period;

int tag = (rank % 2 == 0) ? 1 : 0;;

if (rank == 0) {

rank\_per\_gran = size\_for\_proc - 1;

rank\_per\_period = 1;

}

else {

rank\_per\_gran = 0;

rank\_per\_period = size\_for\_proc - 2;

}

cout << "Proc " << rank << " start = " << start\_in\_proc << " end = " << end\_in\_proc << " size = " << size\_for\_proc << " rank\_per\_gran = " << rank\_per\_gran << " rank\_per\_period = " << rank\_per\_period << endl;

double\* x\_array\_loc = new double[size\_for\_proc];

for (int i = 0; i < size\_for\_proc; i++) {

x\_array\_loc[i] = -PI \* R + (i + start\_in\_proc) \* step\_x;

}

double\*\* u\_loc = new double\* [2];

for (int j = 0; j < 2; j++) {

u\_loc[j] = new double[size\_for\_proc];

for (int i = 0; i < size\_for\_proc; i++) {

if (j == 0) {

u\_loc[j][i] = phi(x\_array\_loc[i]);

}

else {

u\_loc[j][i] = 0.0;

}

}

}

int nubmer\_new = 1;

int nubmer\_old = 0;

int temp = 0;

double u\_gran = 0.0;

double u\_period = 0.0;

clock\_t start\_time\_parallel = clock();

for (int k = 0; k < K; k++) {

/\*nubmer\_new = (k % 2 == 0) ? 1 : 0;

nubmer\_old = (k % 2 != 0) ? 1 : 0;\*/

MPI\_Send(&u\_loc[nubmer\_old][rank\_per\_gran], 1, MPI\_DOUBLE, tag, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&u\_gran, 1, MPI\_DOUBLE, tag, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Send(&u\_loc[nubmer\_old][rank\_per\_period], 1, MPI\_DOUBLE, tag, 0, MPI\_COMM\_WORLD);

MPI\_Recv(&u\_period, 1, MPI\_DOUBLE, tag, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

if (rank == 0) {

u\_loc[nubmer\_new][size\_for\_proc - 1] = u\_loc[nubmer\_old][size\_for\_proc - 1] + gamma \* (u\_gran - 2 \* u\_loc[nubmer\_old][size\_for\_proc - 1] + u\_loc[nubmer\_old][size\_for\_proc - 1 - 1]);

u\_loc[nubmer\_new][0] = (u\_loc[nubmer\_old][1] + u\_period) / 2;

}

else {

u\_loc[nubmer\_new][0] = u\_loc[nubmer\_old][0] + gamma \* (u\_loc[nubmer\_old][1] - 2 \* u\_loc[nubmer\_old][0] + u\_gran);

u\_loc[nubmer\_new][size\_for\_proc - 1] = (u\_period + u\_loc[nubmer\_old][size\_for\_proc - 2]) / 2;

}

for (int i = 1; i < size\_for\_proc - 1; i++) {

u\_loc[nubmer\_new][i] = u\_loc[nubmer\_old][i] + gamma \* (u\_loc[nubmer\_old][i + 1] - 2 \* u\_loc[nubmer\_old][i] + u\_loc[nubmer\_old][i - 1]);

}

temp = nubmer\_new;

nubmer\_new = nubmer\_old;

nubmer\_old = temp;

}

MPI\_Barrier(MPI\_COMM\_WORLD);

if (rank == 0) {

clock\_t end\_time\_parallel = clock();

double duration\_parallel = static\_cast<double>(end\_time\_parallel - start\_time\_parallel) / CLOCKS\_PER\_SEC;

cout << " Time - >" << duration\_parallel << endl;

}

string name\_file\_x\_array = "E:\\study\\Мага\\Параллельные\\arrays\\x\_array\_MPI\_"+to\_string(rank) + extension;

string name\_file\_u\_loc = "E:\\study\\Мага\\Параллельные\\arrays\\u\_t\_array\_MPI\_"+to\_string(rank) + extension;

writeArray(name\_file\_x\_array, x\_array\_loc, size\_for\_proc);

writeArray(name\_file\_u\_loc, u\_loc[nubmer\_new], size\_for\_proc);

delete[] x\_array\_loc;

for (int j = 0; j < 2; j++) {

delete[] u\_loc[j];

}

delete[] u\_loc;

MPI\_Finalize();

}

return 0;

}