Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе №2**

**по дисциплине «Параллельное программирование»**

**«Реализовать заданный метод численного интегрирования на языке С++ с использованием стандарта OpenMP.»**

Выполнили

студенты группы АВТ-813:

Кинчаров Данил

Пайхаев Алексей

Чернаков Кирилл

Преподаватель:

Ландовский Владимир Владимирович,

к.т.н., доцент кафедры АСУ

г. Новосибирск

2020 г.

Содержание

[1. Описание алгоритма. 3](#_Toc50674855)

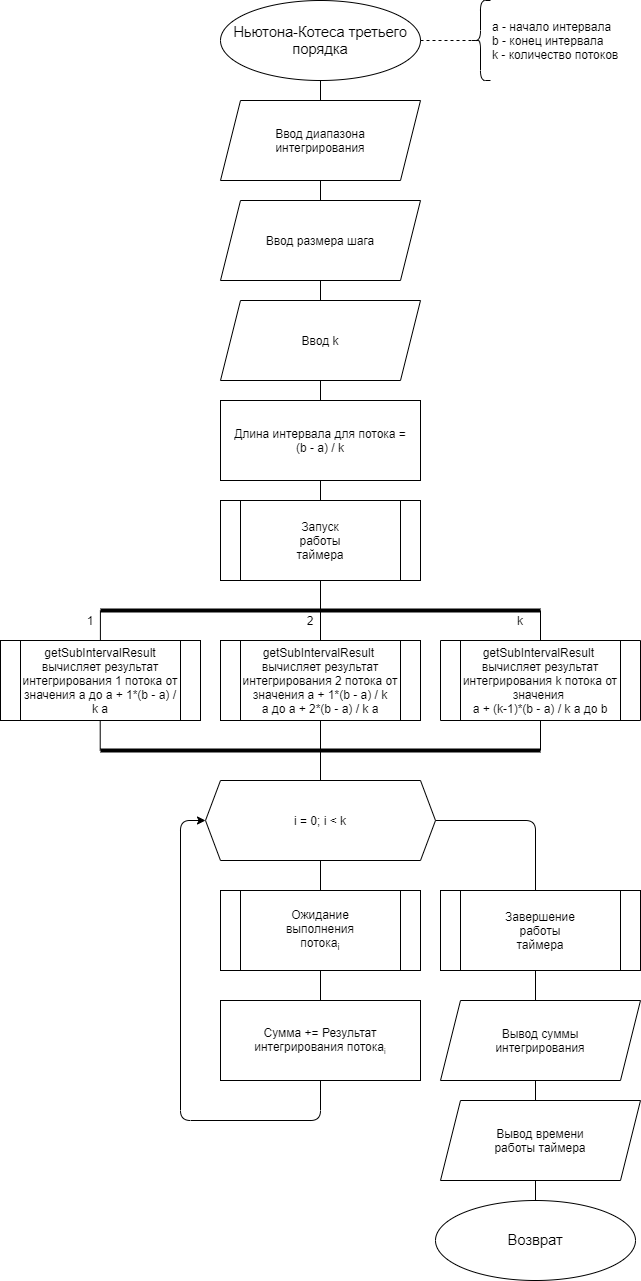
[2. Текст программы. 4](#_Toc50674856)

[3. Примеры работы программы. 4](#_Toc50674857)

[4. Результаты экспериментов. 7](#_Toc50674858)

[5. Выводы. 8](#_Toc50674859)

# **Описание алгоритма.**



# **2. Текст программы.**

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include "pthread.h"

#include <conio.h>

#include <string>

#include <chrono>

#include <vector>

#include <omp.h>

using namespace std;

void scanDoubleWithMessage(double\*, string);

void scanThreadsNumber(int\*);

int getIntervalsNumber(double, double, double);

double getThirdOrderSum(double, double);

void\* getSubIntervalResult(void\*);

double integrationFunction(double);

double thirdOrderNewtonCotesIntegral(int, double, double, double);

void driver();

int main()

{

do {

driver();

} while (\_getch() != EOF);

}

void driver() {

int threadsNumber = 1;

double left = 0, right = 0, step = 0;

scanDoubleWithMessage(&left, "Enter left integration limit: ");

scanDoubleWithMessage(&right, "Enter right integration limit: ");

scanDoubleWithMessage(&step, "Enter integration step: ");

scanThreadsNumber(&threadsNumber);

auto start = std::chrono::system\_clock::now();

cout << "Result: " << thirdOrderNewtonCotesIntegral(threadsNumber, left, right, step) << endl;

auto end = std::chrono::system\_clock::now();

cout << "Time: " << std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() << endl;

}

double thirdOrderNewtonCotesIntegral(int threadsNumber, double left, double right, double step) {

double result = 0;

int intervalsNumber = 0, thirdOrderCn = 8, subIntervalSize = 0;

vector<pthread\_t> threads = {};

intervalsNumber = getIntervalsNumber(left, right, step);

#pragma omp parallel for num\_threads(threadsNumber) reduction(+:result)

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

result += getThirdOrderSum(left + step \* i, step / 3);

}

result \*= step / thirdOrderCn;

return result;

}

double getThirdOrderSum(double left, double step) {

vector<int> thirdOrderTable = { 1, 3, 3, 1 };

double sum = 0;

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

sum += integrationFunction(left + i \* step) \* thirdOrderTable[i];

}

return sum;

}

int getIntervalsNumber(double left, double right, double step) {

int intervalsNumber = ceil(abs(right - left) / step);

return intervalsNumber;

}

double integrationFunction( double x ) {

return 1 / (sqrt(pow(x, 3) + 1));

}

void scanDoubleWithMessage(double\* number, string message) {

cout << message;

cin >> \*number;

}

void scanThreadsNumber(int\* threadsNumber) {

cout << "Enter threads number from 1 to 8: ";

cin >> \*threadsNumber;

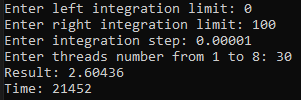
if (\*threadsNumber < 1 || \*threadsNumber > 8) {

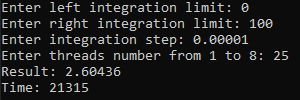
\*threadsNumber = 1;

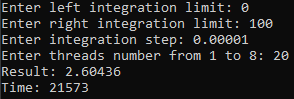
}

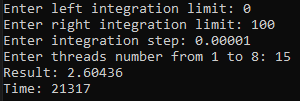
}

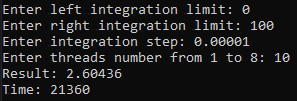
# **3. Примеры работы программы.**

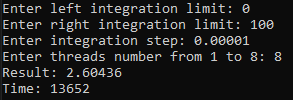


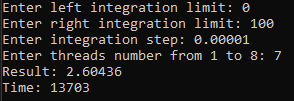


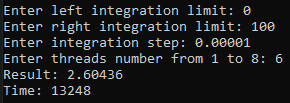


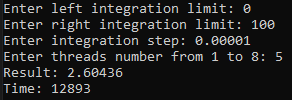


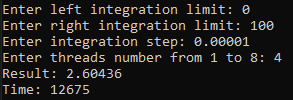


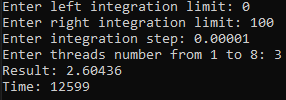


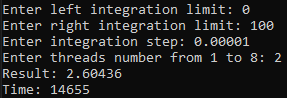


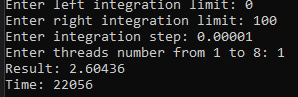












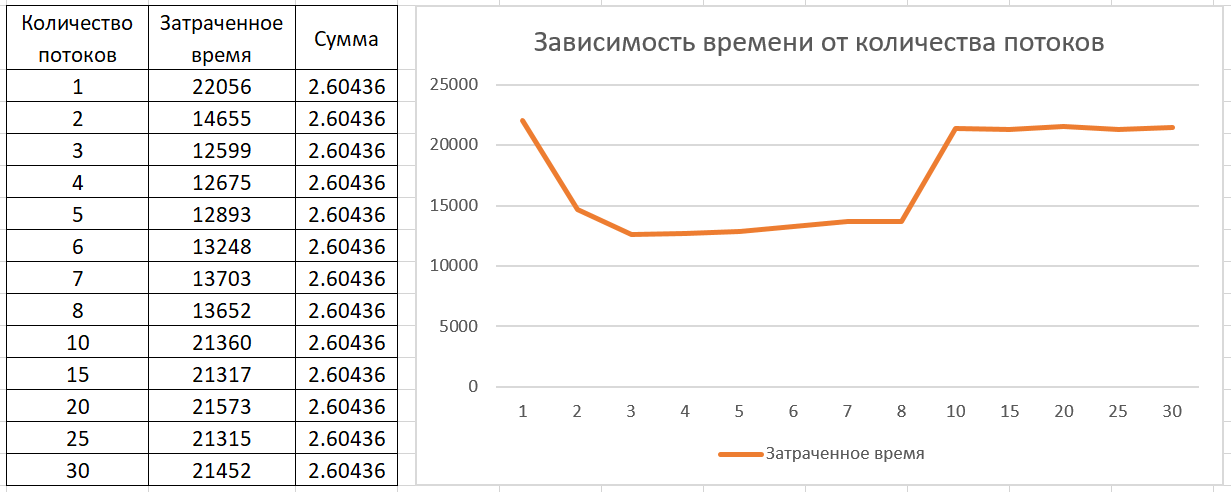
# **4. Результаты экспериментов.**

Подынтегральная функция:

Пределы интегрирования:

Шаг интегрирования:

В таблице приведены результаты выполнения программы, выполнявшейся с использованием процессора 4/4, а также график



# **5. Выводы.**

Основная идея "инкрементального распараллеливания" OpenMP идеально подходит для быстрого распараллеливания вычислительной программы с циклами c большим количеством независимых итераций.

При увеличении количества потоков время работы программы уменьшается. Потоки дают прирост вычислительной мощности только тогда, когда процессор может параллельно работать с всеми этими потоками. Исходя из таблицы и графика можно сказать, что на процессоре с 4 потоками время выполнения уменьшалось, пока количество потоков в программе не достигло 4. Если и дальше увеличивать число потоков, то будет наблюдаться увеличение времени выполнения из-за затрат на их создание и переключение между ними.