Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования «Кубанский государственный университет»

Кафедра вычислительных технологий

**ОТЧЕТ**

о выполнении лабораторной работы № 3

по дисциплине Теория параллельных алгоритмов

Тема: Численное интегрирование по формуле Симпсона

Выполнила: ст. гр. 36/1

Фролова А.А.

Проверила: Патыковская М.В.

Краснодар

2024

**Цель работы:**

Построить программу для вычисления интеграла методом Симпсона. Проследить зависимость времени работы от количества потоков и ядер.

**Ход работы.**

Необходимо посчитать определенный интеграл:

Точность: 0,0001.

Разбиение начать с четырех отрезков: 2n = 4.

Оценивание погрешности происходит за счет метода Рунге по формуле:

Для работы будет использовать язык C++ и библиотеку OpenMp.

Данные, полученные при реализации последовательного алгоритма:

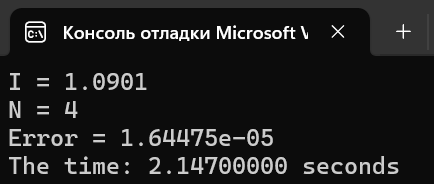


Рисунок 1 – Результат работы последовательного алгоритма

**Возможности для распараллеливания.**

В данной программе можно распараллелить вычисление суммы в циклах for с помощью директивы #pragma omp parallel for reduction(+:sum) и использовать reduction для переменной sum, чтобы избежать гонок данных.

**Параллельная реализация.**

Результат работы на 4 потоках:

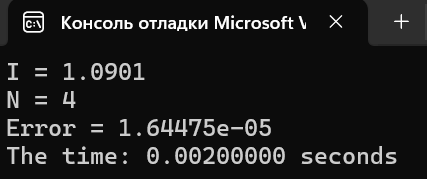


Рисунок 2 – Результат работы параллельного алгоритма на 4 потоках

Данные, полученные в результате работы программы с использование OpenMP:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| потоки | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| время | 0,03 | 0,03 | 0,002 | 0,003 | 0,003 | 0,005 | 0,015 |

График зависимости времени работы от количества потоков:

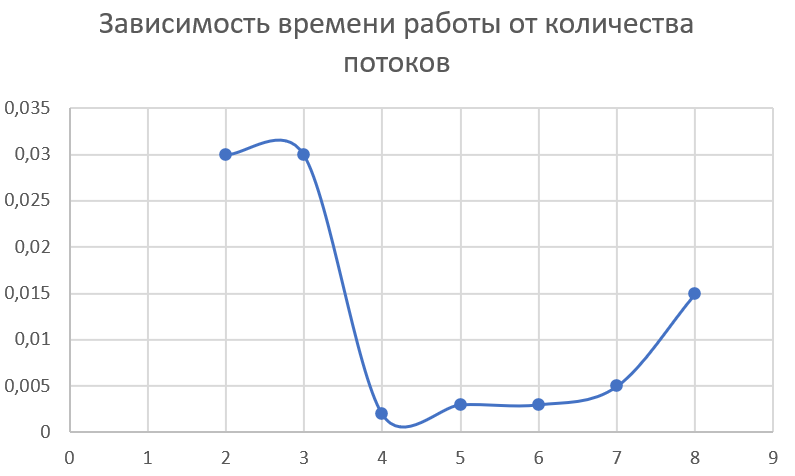
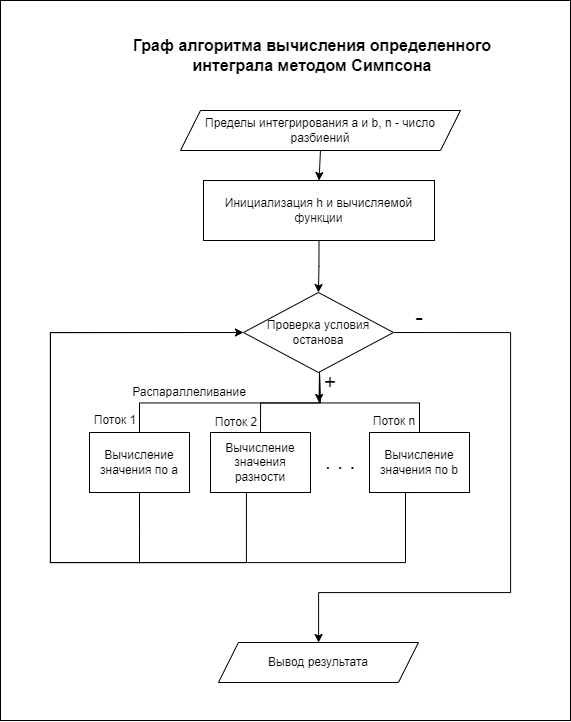


Рисунок 3 – Диаграмма зависимости времени работы программы от количества ядер и размерности системы

На диаграмме представлена зависимости времени работы программы от количества потоков. Можно заметить, что при увеличении размерности системы, увеличивается и время работы алгоритма. Оптимальное время достигается тогда, когда количество потоков равно 4. Это связано с количеством ядер компьютера, на котором производились вычисления (4 ядра).

**Граф алгоритма**

****

**Листинг программ**

Последовательная реализация:

#include <iostream>

#include <cmath>

using namespace std;

double func(double x) {

// f(x)

return sqrt(1 + 2 \* pow(x, 2) - pow(x, 3));

}

double simpson(double a, double b, int n) {

double h = (b - a) / n;

double sum = func(a) + func(b);

for (int i = 1; i < n; i += 2) {

sum += 4 \* func(a + i \* h);

}

for (int i = 2; i < n - 1; i += 2) {

sum += 2 \* func(a + i \* h);

}

return sum \* h / 3;

}

int main() {

double a = 0.0; // Нижний предел интегрирования

double b = 1.0; // Верхний предел интегрирования

double e = 0.0001; // Требуемая точность

int n = 4; // Начальное количество отрезков

clock\_t start = clock();

double integral\_prev = simpson(a, b, n);

double integral\_cur = simpson(a, b, 2 \* n);

long double error = std::abs(integral\_cur - integral\_prev) / 15; // Формула Рунге

while (error > tol) {

n \*= 2;

integral\_prev = integral\_cur;

integral\_cur = simpson(a, b, 2 \* n);

error = std::abs(integral\_cur - integral\_prev) / 15;

}

clock\_t end = clock();

long double seconds = (long double)(end - start) / CLOCKS\_PER\_SEC;

std::cout << "I = " << integral\_cur << std::endl;//интеграл

std::cout << "N = " << n << std::endl;//кол-во отрезков

std::cout << "Error = " << error << std::endl;//погрешность

printf("The time: %.8f seconds\n", seconds);//время работы

return 0;

}

Параллельная реализация:

#include <iostream>

#include <cmath>

#include <omp.h>

using namespace std;

double func(double x) {

// f(x)

return sqrt(1 + 2 \* pow(x, 2) - pow(x, 3));

}

double simpson(double a, double b, int n) {

double h = (b - a) / n;

double sum = func(a) + func(b);

//#pragma omp parallel for reduction(+:sum)

#pragma omp\_set\_num\_threads(8)

for (int i = 1; i < n; i += 2) {

sum += 4 \* func(a + i \* h);

}

//#pragma omp parallel for reduction(+:sum)

#pragma omp\_set\_num\_threads(8)

for (int i = 2; i < n - 1; i += 2) {

sum += 2 \* func(a + i \* h);

}

return sum \* h / 3;

}

int main() {

clock\_t start = clock();

double a = 1.2; // Нижний предел интегрирования

double b = 2.0; // Верхний предел интегрирования

double tol = 0.0001; // Требуемая точность

int n = 4; // Начальное количество отрезков

double integral\_prev = simpson(a, b, n);

double integral\_cur = simpson(a, b, 2 \* n);

long double error = std::abs(integral\_cur - integral\_prev) / 15; // Формула Рунге

#pragma omp\_set\_num\_threads(8)

while (error > tol) {

n \*= 2;

integral\_prev = integral\_cur;

integral\_cur = simpson(a, b, 2 \* n);

error = std::abs(integral\_cur - integral\_prev) / 15;

}

std::cout << "I = " << integral\_cur << std::endl;//интеграл

std::cout << "N = " << n << std::endl;//кол-во отрезков

std::cout << "Error = " << error << std::endl;//погрешность

clock\_t end = clock();

long double seconds = (long double)(end - start) / CLOCKS\_PER\_SEC;

printf("The time: %.8f seconds\n", seconds);//время работы

return 0;

}

Вывод:

В результате выполнения лабораторной работы №3 были созданы программы для вычисления определенного интеграла методом Симпсона последовательно и параллельно. В ходе работы было установлено, что оптимальное количество потоков было примерно равно количеству ядер компьютера, на котором производились расчеты – 4 ядер. Наглядно продемонстрировали, что, используя OpenMP, 4 узла эффективнее, чем 8 узлов. В дальнейшем полученные знания будут использованы для написания более сложных программ.