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»

Выполнил:

Корнилов К.А.

Проверил: преподаватель

Нигодин Е. А.

Краснодар

2024

**Тема работы:** Численное интегрирование по формуле Симпсона.

**Цель:** Распараллеливание процесса численного интегрирования по формуле Симпсона с использованием технологии OpenMP.

**Задание:**

1. Составить последовательный алгоритм. Построить граф алгоритма. Исследовать на нём зависимости и возможности распараллеливания.
2. Выполнить параллельную реализацию алгоритма
3. Составить таблицу, отражающую сравнительное время выполнения последовательного и параллельных алгоритмов для разных размеров матриц на 2-х и 4-х ядрах. (+ 3 ядра, + график в Excel).

**Ход работы:**

1. Был составлен и написан на языке С++ последовательный алгоритм численного интегрирование по формуле Симпсона.

Входными данными алгоритма выступают: функция f(x), для которой вычисляется интеграл, интервал интегрирование (a;b), количество отрезков, на которые разбивается интервал интегрирования.

Результатом выступает приближенное значение определенного интеграла функции f(x) на интервале (a;b).

Данное приближенное значение интеграла вычисляется путем суммирования значений интеграла на отрезках [], вычисленных следующим образом:

Тогда последовательный алгоритм вычисления численного интеграла по формуле Симпсона схематично изображен на рисунке 1 для частного случая (4 отрезков):

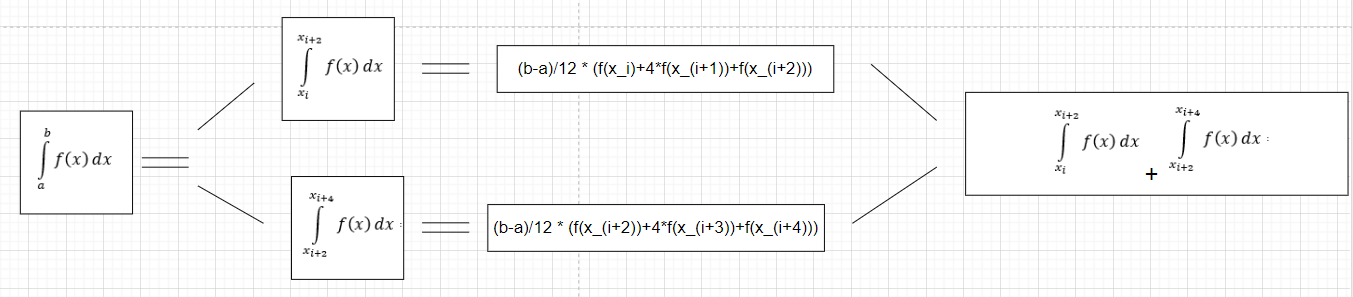


Рисунок 1 – Схема последовательного алгоритма вычисления численного интеграла по формуле Симпсона

Код реализованного алгоритма имеет следующий вид:

double Simpson(double a, double b, double N) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double res = 0;

for (int i = 0; i < N-1; i=i+2) {

res += step / 3 \* (func\_to\_calc(X\_int[i]) + 4 \* func\_to\_calc((double)(X\_int[i + 1])) + func\_to\_calc(X\_int[i + 2]));

}

return res;

}

Согласно схеме алгоритма первой возможностью распараллеливания является распараллеливание вычисления значений интеграла на каждом из отрезков. Каждая строка вычисления значения интеграла итерации может быть выполнена не зависимо от других строк. Результаты этих вычислений затем могут быть сложены в 1 результирующее значение. Схема алгоритма параллельного алгоритма вычисления численного интегрерирования представлена на рисунке 2.

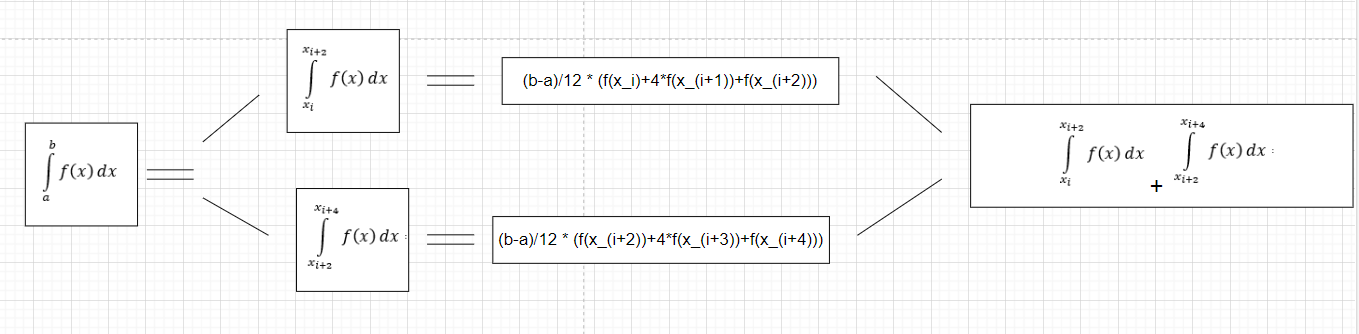


Рисунок 2 – Схема алгоритма параллельного алгоритма вычислений k+1 итерации для алгоритма Якоби решения систем линейных уравнений

Кроме того, можно таким же образом поступить с вычислением значений функций для подсчета интеграла на очередном интервале. Данное действие можно распараллелить, а затем использовать результат для формирование конечного значения интеграла на данном интервале. Схема второго варианта параллельного алгоритма вычисления численного интегрирования по формуле Симпсона представлена на рисунке 3.

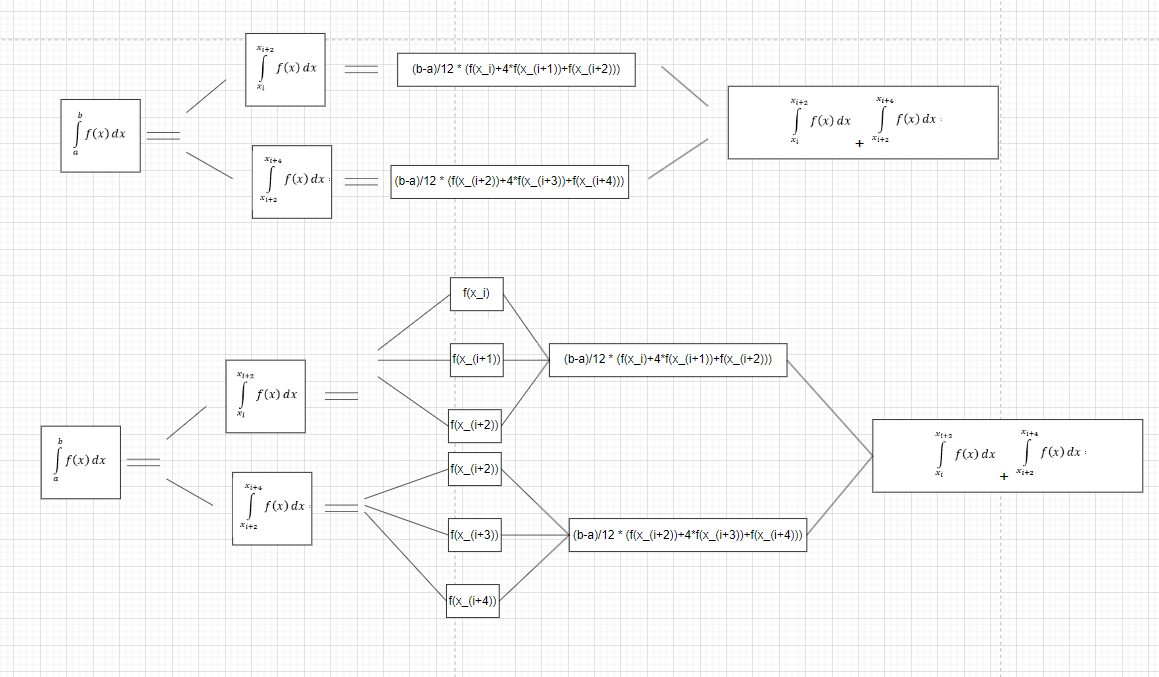


Рисунок 3 – Схема второго варианта второго варианта параллельного алгоритма вычисления численного интегрирования по формуле Симпсона

1. Была написана параллельная реализация численного интегрирования по формуле Симпсона на языке С++ с использованием технологии OpenMP.

Код первого варианта параллельной реализации численного интегрирования по формуле Симпсона:

double Simpson\_parallel(double a, double b, int N,int num\_thr) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double res = 0;

#pragma omp parallel num\_threads(num\_thr) reduction(+:res)

{

#pragma omp for collapse(1)

for (int i = 0; i < N - 1; i = i + 2)

{

res += step / 3 \* (func\_to\_calc(X\_int[i]) + 4 \* func\_to\_calc((double)(X\_int[i + 1])) + func\_to\_calc(X\_int[i + 2]));

}

};

return res;

}

Код второго варианта параллельной реализации численного интегрирования по формуле Симпсона:

double Simpson\_parallel\_2(double a, double b, int N,int num\_thr) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double\* res\_f = new double[3];

double res = 0;

#pragma omp parallel num\_threads(num\_thr) reduction(+:res)

{

#pragma omp for collapse(2)

for (int i = 0; i < N - 1; i = i + 2)

{

for (int j = 0; j < 3; j++) {

res\_f[j] = func\_to\_calc(X\_int[i + j]);

}

res += step / 3 \* (res\_f[0] + 4 \* res\_f[1] + res\_f[2]);

}

};

return res;

}

1. Используя написанные программы, была проведена оценка времени работы алгоритмов для различного количества отрезков и для различного количество ядер. Результаты выполненных оценок представлены в виде графика времени выполнения на рисунке 4.

По оси Ox изображены количества отрезков для разбиения. По оси Oy изображено время работы алгоритма.

Голубым цветом изображено время выполнения для последовательного алгоритма.

Оранжевым изображено время выполнения для первого параллельного алгоритма на 2 ядрах.

Светло–оранжевым изображено время выполнения для первого параллельного алгоритма на 3 ядрах.

Желтым изображено время выполнения для первого параллельного алгоритма на 4 ядрах.

Зеленым изображено время выполнения для второго параллельного алгоритма на 2 ядрах.

Светло–зеленым изображено время выполнения для второго параллельного алгоритма на 3 ядрах.

Темно–зеленым изображено время выполнения для второго параллельного алгоритма на 4 ядрах.

Рисунок 4 – График зависимости времени работы программы от размера системы и количества ядер

**Вывод:** Было осуществлено распараллеливание процесса вычисления приближенного значения интеграла по формуле Симпсона с использованием технологии OpenMP и была осуществлена оценка времени работы данной программы. Согласно проведенными исследованиям максимальная скорость работы программы была достигнута на 2 потоках.

**ПРИЛОЖЕНИЕ.**

**Приложение 1. Полный код программы**

#include<iostream>

#include<string>

#include<sstream>

#include<omp.h>

using namespace std;

double func\_to\_calc(double x) {

return x\*x;

}

double Simpson(double a, double b, double N) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double res = 0;

for (int i = 0; i < N-1; i=i+2) {

res += step / 3 \* (func\_to\_calc(X\_int[i]) + 4 \* func\_to\_calc((double)(X\_int[i + 1])) + func\_to\_calc(X\_int[i + 2]));

}

return res;

}

double Simpson\_parallel(double a, double b, int N,int num\_thr) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double res = 0;

#pragma omp parallel num\_threads(num\_thr) reduction(+:res)

{

#pragma omp for collapse(1)

for (int i = 0; i < N - 1; i = i + 2)

{

res += step / 3 \* (func\_to\_calc(X\_int[i]) + 4 \* func\_to\_calc((double)(X\_int[i + 1])) + func\_to\_calc(X\_int[i + 2]));

}

};

return res;

}

double Simpson\_parallel\_2(double a, double b, int N,int num\_thr) {

double step = (double)(b - a) / N;

double\* X\_int = new double[N + 1];

X\_int[0] = a;

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

X\_int[i] = X\_int[i - 1] + step;

}

double\* res\_f = new double[3];

double res = 0;

#pragma omp parallel num\_threads(num\_thr) reduction(+:res)

{

#pragma omp for collapse(2)

for (int i = 0; i < N - 1; i = i + 2)

{

for (int j = 0; j < 3; j++) {

res\_f[j] = func\_to\_calc(X\_int[i + j]);

}

res += step / 3 \* (res\_f[0] + 4 \* res\_f[1] + res\_f[2]);

}

};

return res;

}

int main()

{

setlocale(LC\_ALL, "russian");

double a,b;

int N;

cout << "Начало отрезка" << endl;

cin >>a;

cout << "Конец отрезка" << endl;

cin >> b;

cout << "Количество отрезков для разбиения" << endl;

cin >> N;

double start\_time = clock();

cout << Simpson\_parallel\_2(a, b, N, 2) << endl;

double end\_time = clock();

cout <<"Второй вид 2 ядра:"<<end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel(a, b, N, 2) << endl;

end\_time = clock();

cout << "Первый вид 2 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel\_2(a, b, N, 3) << endl;

end\_time = clock();

cout << "Второй вид 3 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel(a, b, N, 3) << endl;

end\_time = clock();

cout << "Первый вид 3 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel\_2(a, b, N,4)<<endl;

end\_time = clock();

cout << "Второй вид 4 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson\_parallel(a, b, N,4)<<endl;

end\_time = clock();

cout << "Первый вид 4 ядра:" << end\_time - start\_time << endl;

start\_time = clock();

cout << Simpson(a, b, N) << endl;

end\_time = clock();

cout << "Последовательный вид:" << end\_time - start\_time << endl;

}