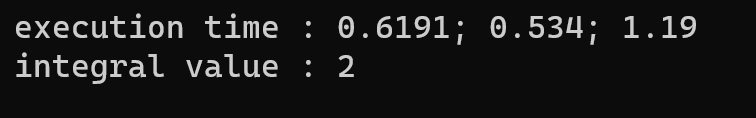
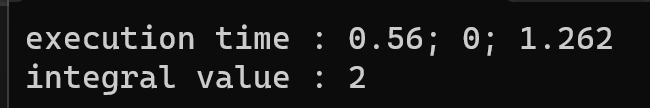
Work 1 Кириленко Константин



Режим Debug

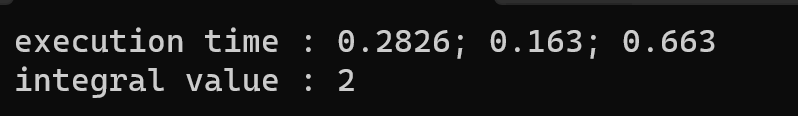


Режим Release



Подобран шаг h = 0.0000001

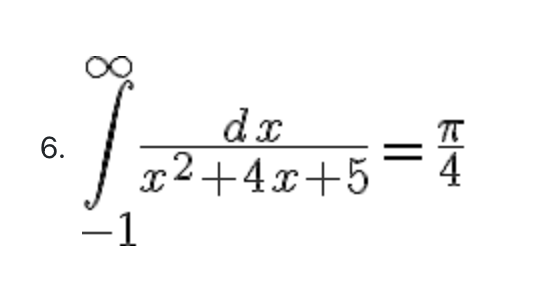
Private reduct



Эти параметры нужны для правильной работы параллельных потоков. Т.е. цикл теперь выполняется не последовательно, а параллельно, и если эти параметры не использовать, то в память хаотично будет записываться результат, и его конечное значение будет неправильным. Вместо параметра private, можно просто сделать x локальной переменной внутри цикла. А параметр reduction помогает подсчитать сумму, после выполнения всех потоков.

Ускорение в 1,98 раза

Формула Симпсона Вариант 6



// 01\_seq.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include <omp.h>

using namespace std;

double f(double x) {

return 1 / (x \* x + 4 \* x + 5);

//return 1 / sqrt(x - x \* x);

}

void integral(const double a, const double b,

double h, double\* res)

{

long long int i, n;

double sum1, sum2; // локальная переменная для подсчета интеграла

double x; // координата точки сетки

n = (int)((b - a) / h); // количество точек сетки интегрирования

sum1 = 0.0;

sum2 = 0.0;

h = (b - a) / (2 \* n);

#pragma omp parallel for reduction(+: sum1) for (i = 1; i <= n; i++)

{

sum1 += f(a + (2 \* i - 1) \* h);

}

#pragma omp parallel for reduction(+: sum2) for (i = 1; i < n; i++)

{

sum2 += f(a + 2 \* i \* h);

}

\*res = h / 3 \* (f(a) + f(b) + 4 \* sum1 + 2 \* sum2);

}

#define PI 3.1415926535897932384626433832795

double experiment(double\* res)

{

double stime, ftime; // время начала и конца расчета

double a = -1; // левая граница интегрирования

double b = 1000000; // правая граница интегрирования

double h = 0.000001; // шаг интегрирования

stime = clock();

integral(a, b, h, res); // вызов функции интегрирования

ftime = clock();

return (ftime - stime) / CLOCKS\_PER\_SEC;

}

int main()

{

int i; // переменная цикла

double time; // время проведенного эксперимента

double res; // значение вычисленного интеграла

double min\_time; // минимальное время работы

// реализации алгоритма

double max\_time; // максимальное время работы

// реализации алгоритма

double avg\_time; // среднее время работы

// реализации алгоритма

int numbExp = 10; // количество запусков программы

// первый запуск

min\_time = max\_time = avg\_time = experiment(&res);

// оставшиеся запуски

for (i = 0; i < numbExp - 1; i++)

{

time = experiment(&res);

avg\_time += time;

if (max\_time < time) max\_time = time;

if (min\_time > time) min\_time = time;

}

// вывод результатов эксперимента

cout << "execution time : " << avg\_time / numbExp << "; " <<

min\_time << "; " << max\_time << endl;

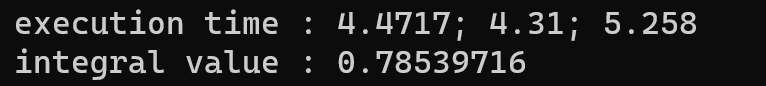
cout.precision(8);

cout << "integral value : " << res << endl;

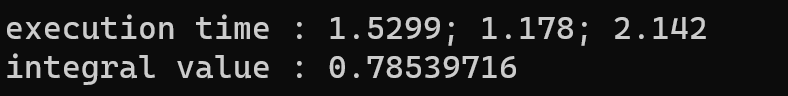
return 0;

}

без директивы



с директивой



Ускорение при использование директивы parallel составило 2,92287 раза