МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ. OPENMP. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ, ФОРМУЛА СИМПСОНА, ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННЫХ И КРАТНЫХ ИНТЕГРАЛОВ.**

ЛАБОРАТОРНАЯ РАБОТА №1

(Вариант №4)

студента 4 курса 451 группы

направления 09.03.04 — Программная инженерия

факультета КНиИТ

Голикова Артема Олеговича

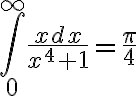
Проверил

канд. физ-мат. н., доцент \_\_\_\_\_\_\_\_\_\_\_\_ К. П. Савина

Саратов 2020

**Цель работы:** изучить использование потоков OpenMP (директивы #pragma omp paralel) и операции редукции, приводящих к уменьшению вычисления интегралов, на примере методов прямоугольников и Симпсона; сравнить время выполнения алгоритмов численного интегрирования с использованием разных режимов.

**Ход работы:**



Код программы реализующий метод прямоугольников для последовательной реализации:

// Work01 - Вариант 4, последовательная версия, метод прямоугольников

#include <time.h>

#include <iostream>

using namespace std;

void integral(const double a, const double b,

const double h, double\* res)

{

int i, n;

double sum; // локальная переменная для подсчета интеграла

double x; // координата точки сетки

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

sum = 0.0;

for (i = 0; i < n; i++)

{

x = a + i \* h + h / 2.0;

sum += (x / (x\*x\*x\*x + 1)) \* h;

}

\*res = sum;

}

double experiment(double\* res)

{

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

double a = 0.0; // левая граница интегрирования

double b = 1000000.0; // правая граница интегрирования

double h = 0.001; // шаг интегрирования

stime = clock();

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

ftime = clock();

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

}

int main()

{

setlocale(LC\_ALL, "Russian");

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

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 << " Среднее время: " << avg\_time / numbExp << "\n Минимальное время: " <<

min\_time << "\n Максимальное время:" << max\_time << endl;

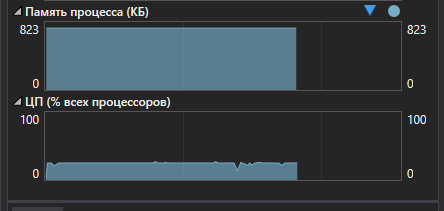
cout.precision(8);

cout << " Значение интеграла: " << res << endl;

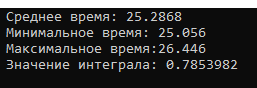
return 0;

}

Использование ЦП во время выполнения:



Результат выполнения:



Код программы реализующий метод прямоугольников для параллельной реализации:

// Work01 - Вариант 4, параллельная версия, метод прямоугольников

#include <time.h>

#include <iostream>

#include <omp.h>

using namespace std;

void integral(const double a, const double b,

const double h, double\* res)

{

int i, n;

double sum; // локальная переменная для подсчета интеграла

double x; // координата точки сетки

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

sum = 0.0;

#pragma omp parallel for private(x) reduction(+: sum)

for (i = 0; i < n; i++)

{

x = a + i \* h + h / 2.0;

sum += (x / (x \* x \* x \* x + 1)) \* h;

}

\*res = sum;

}

double experiment(double\* res)

{

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

double a = 0.0; // левая граница интегрирования

double b = 1000000.0; // правая граница интегрирования

double h = 0.001; // шаг интегрирования

stime = clock();

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

ftime = clock();

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

}

int main()

{

setlocale(LC\_ALL, "Russian");

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

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 << " Среднее время: " << avg\_time / numbExp << "\n Минимальное время: " <<

min\_time << "\n Максимальное время:" << max\_time << endl;

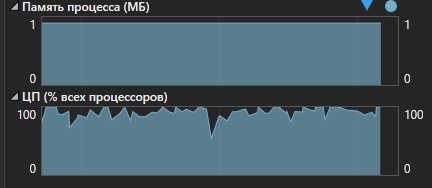
cout.precision(8);

cout << " Значение интеграла: " << res << endl;

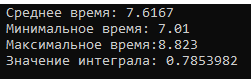
return 0;

}

Использование ЦП во время выполнения:



Результат выполнения:



Код программы реализующий метод Симпсона для последовательной реализации:

// Work01 - Вариант 4, последовательная версия, метод Симпсона

#include <time.h>

#include <iostream>

using namespace std;

void integral(const double a, const double b,

const double h, const int N, double\* res)

{

int i;

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

double x; // координата точки сетки

sum1 = 0.0;

sum2 = 0.0;

for (i = 0; i < N; i++)

{

x = a + (2 \* double(i) - 1) \* h;

sum1 += (x / (x \* x \* x \* x + 1));

}

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

{

x = a + 2 \* double(i) \* h;

sum2 += (x / (x \* x \* x \* x + 1));

}

\*res = (h / 3) \* ((a / (a \* a \* a \* a + 1)) + (b / (b \* b \* b \* b + 1)) + (4 \* sum1) + (2 \* sum2));

}

double experiment(double\* res)

{

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

double a = 0.0; // левая граница интегрирования

double b = 1000000.0; // правая граница интегрирования

double h = 0.001; // шаг интегрирования

int N = ((b - a) / h) / 2;

stime = clock();

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

ftime = clock();

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

}

int main()

{

setlocale(LC\_ALL, "Russian");

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

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 << " Среднее время: " << avg\_time / numbExp << "\n Минимальное время: " <<

min\_time << "\n Максимальное время:" << max\_time << endl;

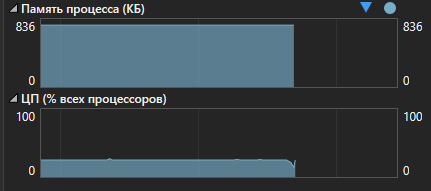
cout.precision(8);

cout << " Значение интеграла: " << res << endl;

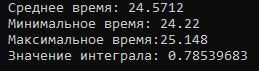
return 0;

}

Использования ЦП во время выполнения:



Результат выполнения:



Код программы реализующий метод Симпсона для параллельной реализации:

// Work01 - Вариант 4, параллельная версия, метод Симпсона

#include <time.h>

#include <iostream>

#include <omp.h>

using namespace std;

void integral(const double a, const double b,

const double h, const int N, double\* res)

{

int i;

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

double x; // координата точки сетки

sum1 = 0.0;

sum2 = 0.0;

#pragma omp parallel for private(x) reduction(+: sum1)

for (i = 0; i < N; i++)

{

x = a + (2 \* double(i) - 1) \* h;

sum1 += (x / (x \* x \* x \* x + 1));

}

#pragma omp parallel for private(x) reduction(+: sum2)

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

{

x = a + 2 \* double(i) \* h;

sum2 += (x / (x \* x \* x \* x + 1));

}

\*res = (h / 3) \* ((a / (a \* a \* a \* a + 1)) + (b / (b \* b \* b \* b + 1)) + (4 \* sum1) + (2 \* sum2));

}

double experiment(double\* res)

{

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

double a = 0.0; // левая граница интегрирования

double b = 1000000.0; // правая граница интегрирования

double h = 0.001; // шаг интегрирования

int N = ((b - a) / h) / 2;

stime = clock();

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

ftime = clock();

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

}

int main()

{

setlocale(LC\_ALL, "Russian");

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

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 << " Среднее время: " << avg\_time / numbExp << "\n Минимальное время: " <<

min\_time << "\n Максимальное время:" << max\_time << endl;

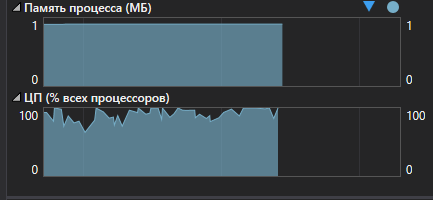
cout.precision(8);

cout << " Значение интеграла: " << res << endl;

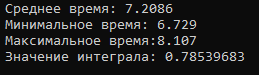
return 0;

}

Использование ЦП во время выполнения:



Результат выполнения:



Сравнение результатов:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Последовательная  реализация | Параллельная  реализация | Ускорение |
| Метод прямоугольников | Мин. время: 25,056  Макс. время: 26,446  Сред. время: 25,2868 | Мин. время: 7,01  Макс. время: 8,823  Сред. время: 7,6167 | 3,3199 |
| Метод Симпсона | Мин. время: 24,22  Макс. время: 25,148  Сред. время: 24,5712 | Мин. время: 6,729  Макс. время: 8,107  Сред. время: 7,2086 | 3,4086 |

Вычисления происходили на процессоре Intel Core i5-2310 (4 ядра, 4 потока). По результатам сравнения видно, что распараллеливание при вычислении интегралов дает прирост по быстродействию в 3,3-3,4 раза.

**Вывод:** изучил использование потоков OpenMP (директивы #pragma omp paralel) и операции редукции, приводящих к уменьшению вычисления интегралов, на примере методов прямоугольников и Симпсона; сравнил время выполнения алгоритмов численного интегрирования с использованием разных режимов.