МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**“УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Факультет информационных систем и технологий

Кафедра “ Вычислительная техника”

Дисциплина “Высокопроизводительные вычисления”

**Лабораторная работа №3**

Исследование многопоточных реализаций алгоритма численного интегрирования в среде одной ПЭВМ

Вариант 3

Выполнил:

студент гр. ИВТАПбд-31

Вершинин Д. В.

Проверил:

Негода В.Н.

Ульяновск, 2019

Цель работы

Изучение методов распараллеливания реализации вычисления определенного интеграла функции, заданной вариантом. Подынтегральная функция ***cos(x).***

**1. Выбор метода численного интегрирования**

В качестве метода численного интегрирования был выбран метод прямоугольников. **Суть метода прямоугольников** заключается в том, что в качестве приближенного значения определенного интеграла берут интегральную сумму. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота — значением подынтегральной функции в этих узлах. Если отрезок интегрирования является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле левых прямоугольников, правых прямоугольников и средних прямоугольников. Наибольшую точность обеспечивает последний метод, поэтому будет использоваться именно он.

Формула средних прямоугольников:

1

Если отрезок интегрирования разбивается на равные части, можно получить составную формулу средних прямоугольников:



**2. Исследование многопоточных реализаций выбранного метода численного интегрирования**

Для реализации многопоточного выполнения цикла вычислений было решено использовать библиотеку OpenMP. Число потоков изменяется от 1 до 8, количество вычислений подынтегральной функции изменяется от 100 до 1000000.

**Конфигурация первого компьютера:**

Ноутбук Lenovo G510

Процессор Intel® Core™ i5-4200M

Количество ядер 2

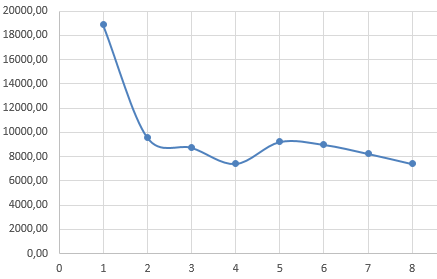
Количество потоков 4

Базовая частота 2.5 Ггц

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | time, мкс | | | | |
| Потоки | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 1.64215 | 18.8851 | 190.904 | 1914.38 | 20192,00 |
| 2 | 1.64215 | 269094,00 | 96.4785 | 958.628 | 11914.1 |
| 3 | 1.23167 | 269094,00 | 98.5316 | 987.367 | 12213,00 |
| 4 | 2.46335 | 7.80029 | 74.3091 | 741.449 | 11366,00 |
| 5 | 2.87383 | 2614712,00 | 107.563 | 1063.73 | 16072.5 |
| 6 | 3.28431 | 11.0846 | 90.7308 | 888.014 | 15283.5 |
| 7 | 3.28431 | 11.0846 | 86.6253 | 848.602 | 17723.3 |
| 8 | 2.87383 | 13.9587 | 103.458 | 743.912 | 19350.7 |

Можно заметить, что минимальное время выполнения кода достигается, когда число порождаемых потоков равно числу потоков управления процессора.

Гранулярность 1000000



**Конфигурация второго компьютера:**

Ноутбук ASUS TUF FX504GM-EN395

Процессор Intel® Core™ i7-8750H

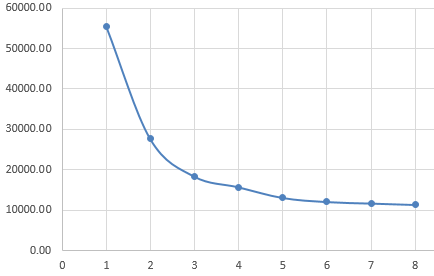
Количество ядер 6

Количество потоков 12

Базовая частота 4.3 Ггц

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | time, мкс | | | | |
| Потоки | 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 5.1 | 50.5 | 505.1 | 5086.6 | 55510.4 |
| 2 | 2.8 | 24.8 | 253 | 2543.7 | 27625.7 |
| 3 | 1.9 | 17.1 | 182.6 | 1693.9 | 18313.2 |
| 4 | 1.6 | 12.8 | 135.6 | 1339.8 | 15682.8 |
| 5 | 1.4 | 10.3 | 101.2 | 1162.9 | 13049.8 |
| 6 | 1.4 | 9.4 | 91.3 | 842.4 | 12037.2 |
| 7 | 1.3 | 7.4 | 73.2 | 829.5 | 11666.2 |
| 8 | 1.4 | 7.1 | 63.5 | 808 | 11324.5 |

Гранулярность 1000000



Для данного компьютера число потоков процессора больше потоков, порождаемых компилятором. В данном случае при увеличении числа потоков время выполнения уменьшается. Минимальное время будет достигаться при 8 потоках.

**3. Вывод**

Анализируя полученные данные, можно сказать, что они подчиняются закону Амдала. С увеличением числа потоков, уменьшается скорость роста производительности. Это связано с тем, что фоновые процессы с более высоким приоритетом занимают потоки и с ростом числа потоков вероятность этого события увеличивается. Иными словами, чем больше потоков задействовано в распараллеливании, тем больше вероятность, что текущую задачу перекроет задача с более высоким приоритетом.

**4. Исходный код**

#include <iostream>

#include <cmath>

#include <omp.h>

int stepMin = 100;

int stepMax = 1000000;

double timing = 0;

double minTiming = 0;

float f(float x) {

return cos(x);

}

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

int i;

double result = 0, h;

h = (b - a) / n;

double startTiming = omp\_get\_wtime();

#pragma omp parallel for reduction(+:result)

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

result += f(a + h \* i + h \* 0.5);

result \*= h;

double endTiming = omp\_get\_wtime();

timing = (endTiming - startTiming);

return result;

}

void singleTest(int maxThreads, int n, int precise) {

std::cout << "steps: " << n;

std::cout << "\nthreads timing\n";

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

omp\_set\_num\_threads(i);

timing = 0;

minTiming = INT32\_MAX;

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

CalcIntegral(0.0F, 1.0F, n);

if (timing < minTiming)

minTiming = timing;

}

std::cout << " " << i << " " << minTiming \* 1000000 << std::endl;

}

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

}

void multipleTests(int maxThreads) {

for (int n = stepMin, precise = 10000; n <= stepMax; n \*= 10, precise /= 10) {

singleTest(maxThreads, n, precise);

}

}

int main() {

multipleTests(8);

//singleTest(8, 1000000, 10);

return 0;

}