

Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра автоматизации систем вычислительных комплексов

Практикум по курсу

«Суперкомпьютеры и параллельная обработка данных»

**Отчет по заданию №1**

Разработка параллельной версии программы для вычисления определенного интеграла методом Симпсона с использованием технологии OpenMP

Коваленко Анастасия Павловна, 321 группа

Москва, 2020

Оглавление

[**Постановка задачи** 3](#_Toc57065682)

[**Текст программы** 3](#_Toc57065683)

[**Результаты измерений времени выполнения** 5](#_Toc57065684)

[**Вывод** 6](#_Toc57065685)

# 

# **Постановка задачи**

Требуется разработать параллельную версию программы, которая вычисляет определенный интеграл методом Симпсона с использованием технологии OpenMP, и исследовать масштабируемость полученной параллельной программы, построив график зависимости времени выполнения от числа нитей для различного объема входных данных.

# **Текст программы**

#include <iostream>

#include <sys/time.h>

#include <sstream>

#include <omp.h>

double fun(double x)

{

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

}

double integral(double a, double b, int n)

{

double res = 0;

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

double x1, x2;

#pragma omp parallel for shared(h, n) private(x1, x2) reduction(+:res) schedule(static)

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

x1 = a + (2 \* i - 1) \* h;

x2 = a + 2 \* h \* i;

res += 4 \* fun(x1);

res += 2 \* fun(x2);

}

res += fun(b - h);

res += fun(a) + fun(b);

res \*= (h / 3);

return res;

}

int main(int argc, char \*\*argv)

{

struct timeval start, finish, diff;

unsigned int n, num\_threads;

std::stringstream s1;

s1 << argv[1];

s1 >> num\_threads;

omp\_set\_num\_threads(num\_threads);

std::stringstream s2;

s2 << argv[2];

s2 >> n;

double a = 0, b = 200;

gettimeofday(&start, NULL);

std::cout << "The result is " << integral(a, b, n) << std::endl;

gettimeofday(&finish, NULL);

diff.tv\_sec = finish.tv\_sec - start.tv\_sec;

diff.tv\_usec = finish.tv\_usec - start.tv\_usec;

if (diff.tv\_usec < 0) {

diff.tv\_sec--;

diff.tv\_usec += 1000000;

}

std::cout << "Time passed: " << diff.tv\_sec << "." << diff.tv\_usec/1000 << "." << diff.tv\_usec%1000 << std::endl;

return 0;

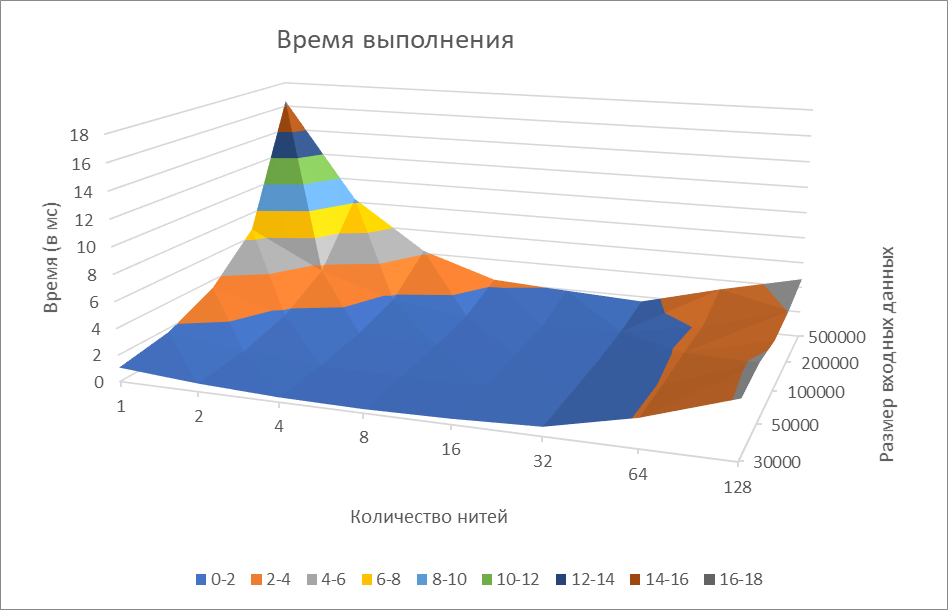
}

# **Результаты измерений времени выполнения**

Ниже приведены результаты измерений времени выполнения полученной программы. Для каждых входных данных программа была запущена несколько раз, в таблице содержатся усредненные результаты без учета случайных выбросов. Время выполнения приведено в формате «секунды.миллисекунды.микросекунды».

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Объем входных данных  Количество  нитей | 30000 | 50000 | 100000 | 200000 | 500000 |
| 1 | 0.1.54 | 0.1.711 | 0.3.374 | 0.6.621 | 0.16.419 |
| 2 | 0.0.607 | 0.0.887 | 0.1.710 | 0.3.619 | 0.8.289 |
| 4 | 0.0.352 | 0.0.510 | 0.0.923 | 0.1.747 | 0.4.210 |
| 8 | 0.0.289 | 0.0.374 | 0.0.652 | 0.1.013 | 0.2.263 |
| 16 | 0.0.408 | 0.0.464 | 0.0.628 | 0.0.854 | 0.1.904 |
| 32 | 0.0.666 | 0.0.711 | 0.0.809 | 0.1.048 | 0.1.605 |
| 64 | 0.2.102 | 0.2.083 | 0.2.193 | 0.2.141 | 0.3.235 |
| 128 | 0.4.179 | 0.4.311 | 0.3.774 | 0.4.039 | 0.4.671 |

Зеленым цветом в таблице выделены ячейки, в которых содержится минимальное время выполнения для каждого набора входных данных.

График зависимости времени исполнения программы от числа нитей и разных входных данных

# **Вывод**

Распараллеливание программы дало ускорение примерно в 10 раз на данных большого размера и около 5-7 раз на данных среднего размера.

В процессе работы были получены следующие результаты:

* Для данных относительно небольшого размера (30 000 – 50 000) оптимальным количеством нитей является 8.
* Для среднего объема данных (100 000-200 000) оптимальное количество нитей – 16. Увеличение накладных расходов на добавление нитей оказывается меньше, чем увеличение объема данных.
* Для работы с данными большого размера (500 000) следует использовать 32 нити. Однако даже при таких входных параметрах накладные расходы оказывают влияние на время выполнения и прирост в скорости оказывается все еще не пропорциональным количеству нитей.