Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Институт прикладной математики и компьютерных наук

ОТЧЕТ

по курсу «Параллельное программирование»

Выполнил студент группы №932201

В. А. Викторов

Проверил старший преподаватель ММФ

В. И. Лаева

Томск-2024

# Задание 6.

# Используя OpenMP, реализовать программу для вычисления определенного интеграла от функции 𝑓(𝑥)= с точностью ε = 107 на отрезке [0.4; 9] с использованием метода правых треугольников. Провести параллельное вычисление интеграла на нескольких процессах и сравнить результаты.

Программа написана на языке C++ с использованием библиотеки OpenMP для параллельных вычислений. В программе используется функция f(x) для вычисления значения подынтегральной функции. Затем интеграл вычисляется методом правых треугольников на каждом процессе, результаты суммируются и выводятся на экран.

Приведём код написанной программы на языке C++:

#include <iostream>

#include <cmath>

#include <iomanip>

#include <omp.h>

using namespace std;

double f(double x) {

return sqrt(x \* (3 + x)) / (x + 1);

}

int main() {

double ans = 8.956794683657;

const double a = 0.4, b = 9;

double I = 0;

unsigned long long n = 10000000;

double h = (b - a) / n;

int num\_threads = omp\_get\_max\_threads();

double start\_time = omp\_get\_wtime();

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

{

double local\_I = 0;

#pragma omp for schedule(static)

for (unsigned long long i = 0; i < n; i++) {

local\_I += f(a + (i + 1) \* h);

}

I += local\_I;

}

I \*= h;

double end\_time = omp\_get\_wtime();

double time\_multi\_thread = end\_time - start\_time;

double time\_single\_thread = 0.165619134903;

double speedup = time\_single\_thread / time\_multi\_thread;

double efficiency = speedup / num\_threads;

cout << num\_threads << endl;

cout << "Result: " << setprecision(13) << I << endl;

cout << setprecision(13) << fabs(ans - I) << endl;

cout << "Time: " << time\_multi\_thread << " seconds" << endl;

cout << "Speedup: " << speedup << endl;

cout << "Efficiency: " << efficiency << endl;

return 0;

}

После выполнения программы на 4 процессах был получен результат:

Result: 8.956794773685

Time: 0.1161539554596

Вывод: Программа успешно реализует параллельное вычисление определенного интеграла с использованием MPI и метода правых треугольников. Проведенные вычисления показывают высокую точность и эффективность распараллеливания задачи.

Приведем результаты расчета программы для *n*  100000:

Количество процессоров size = 1

integral = 8.956794773685 time = 0.16593

Количество процессоров size = 2

integral = 8.956794773685 time = 0.08492

Количество процессоров size = 4

integral = 8.956794773685 time = 0 0.11615

Количество процессоров size = 5

integral = 8.956794773685 time = 0.06829

Количество процессоров size = 10

integral = 8.956794773685 time = 0.05474

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение

*SP*  *T*1 / *Tp* и эффективность

*EP*  *Sp* / *p* :

*S*  *T*1  0.16593  1.95 *E*  *S*2  1.95 0.97

2

2

*T*2 0.08492 2 2

*S*  *T*1  0.16593  1.43 *E*  *S*4  1.43 0.356

4

4

*T*4 0.11615 4 4

*S*  *T*1  0.16593 2.42 *E*  *S*5  2.42 0.48

5

5

*T*5 0.06829 5 5

*S*  *T*1  0.16593 3.02 *E*  *S*10  3.02 0.302

10 0.05474 10 10 10

*T*

10

Программа демонстрирует ускорение и эффективность при увеличении числа процессоров до определенного предела, после чего ускорение уменьшается. Например, при двух процессорах ускорение составляет примерно 1.95, что означает, что программа работает примерно в 1.95 раза быстрее, чем при одном процессоре. Однако при увеличении числа процессоров до 4 ускорение снижается до 1.43, а эффективность до 0.356. Это может быть связано с увеличением накладных расходов на управление процессами и коммуникацию между ними при большем числе процессоров. Но при увеличении числа до 10 ускорение растёт до 3.02, а эффективность до 0.302.