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

Д. А. Прокопьев

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 = 10-10 на отрезке [0; 5.1] 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 <omp.h>

#include <cmath>

#include <iomanip>

using namespace std;

double f(double x) {

// Подынтегральная функция

return (1.4 + sqrt(x + pow(exp(1), x))) / (x + 0.7);

}

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

double ans = 10.0735117837278; // Ожидаемый результат интеграла

const double a = 0, b = 5.1;

const int n = 10000000; // Общее количество подинтервалов

double h = (b - a) / n; // Шаг интегрирования

double total\_integral = 0.0; // Общая сумма интеграла

// Начало времени измерения

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

// Распределение работы между потоками

#pragma omp parallel for reduction(+:total\_integral)

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

double x\_mid = a + (i + 0.5) \* h; // Средняя точка подинтервала

total\_integral += f(x\_mid);

}

total\_integral \*= h; // Умножаем на шаг интегрирования

// Вывод результатов

cout << "Calculated Integral: " << setprecision(13) << total\_integral << endl;

cout << "Difference from expected: " << setprecision(13) << fabs(ans - total\_integral) << endl;

cout << "Time taken: " << omp\_get\_wtime() - start\_time << " seconds" << endl;

return 0;

}

После выполнения программы на 1 процессе был получен результат:

Result: 10.07351178372

Time: 0.3930749893188

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 = 10.07351178372 time = 0.3930749893188

Количество процессоров size = 2

integral = 10.07351178372 time = 0.2035119533539

Количество процессоров size = 4

integral = 10.07351178372 time = 0 0.1087310314178

Количество процессоров size = 5

integral = 10.07351178372 time = 0.1539080142975

Количество процессоров size = 10

integral = 10.07351178372 time = 0.1215319633484

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение

*SP*  *T*1 / *Tp* и эффективность

*EP*  *Sp* / *p* :

*S*  *T*1  0.39307  1.93 *E*  *S*2  1.93 0.965

2

2

*T*2 0.20351 2 2

*S*  *T*1  0.39307  3.61 *E*  *S*4  3.61 0.9025

4

4

*T*4 0.10873 4 4

*S*  *T*1  0.39307 2.55 *E*  *S*5  2.55 0.51

5

5

*T*5 0.15390 5 5

*S*  *T*1 0.39307 3.23 *E*  *S*10  3.23 0.323

10 0.12153 10 10 10

*T*

10

Программа демонстрирует ускорение и эффективность при увеличении числа процессоров до определенного предела, после чего ускорение уменьшается. Например, при трех процессорах ускорение составляет примерно 3.61, что означает, что программа работает примерно в 3.61 раза быстрее, чем при одном процессоре. Однако при увеличении числа процессоров до 4 ускорение снижается до 2.55, а эффективность до 0.51. 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 3.23, а эффективность до 0.323.