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 3.

# Используя MPI, реализовать программу для вычисления определенного интеграла от функции 𝑓(𝑥)= с точностью ε = 10-10 на отрезке [0; 5.1] с использованием метода средних прямоугольников. Провести параллельное вычисление интеграла на нескольких процессах и сравнить результаты.

Программа написана на языке C++ с использованием библиотеки MPI 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 <mpi.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

int rank, size;

const double a = 0, b = 5.1;

const int n = 10000000; // Общее количество подинтервалов

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

double local\_integral = 0.0; // Локальная сумма интеграла

double\* Result = 0;

// Инициализация MPI

MPI\_Init(&argc, &argv);

MPI\_Comm comm = MPI\_COMM\_WORLD;

MPI\_Comm\_size(comm, &size);

MPI\_Comm\_rank(comm, &rank);

if (rank == 0) {

Result = new double[size];

}

// Начало времени измерения

double start\_time = MPI\_Wtime();

// Распределение работы между процессами

for (int i = rank; i < n; i += size) {

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

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

}

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

// Собираем результаты со всех процессов на процесс 0

MPI\_Gather(&local\_integral, 1, MPI\_DOUBLE, Result, 1, MPI\_DOUBLE, 0, comm);

if (rank == 0) {

double total\_integral = 0.0;

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

total\_integral += Result[i];

}

// Вывод результатов

cout << "Number of processes: " << size << endl;

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

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

cout << "Time taken: " << MPI\_Wtime() - start\_time << " seconds" << endl;

delete[] Result;

}

// Завершение MPI

MPI\_Finalize();

return 0;

}

После выполнения программы на 2 процессах был получен результат:

Result: 10.07351178372

Time: 0.2026090621948

Error: 3.696598582792e-12

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*  10000000:

Количество процессоров size = 1

integral = 10.07351178372 time = 0.4044709205627

Количество процессоров size = 2

integral = 10.07351178372 time = 0.2026090621948

Количество процессоров size = 4

integral = 10.07351178372 time = 0 0.1018059253693

Количество процессоров size = 5

integral = 10.07351178372 time = 0.09562397003174

Количество процессоров size = 10

integral = 10.07351178372 time = 0.04828310012817

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение

*SP*  *T*1 / *Tp* и эффективность

*EP*  *Sp* / *p* :

*S*  *T*1  0.40447 2.0 *E*  *S*2  2  1

2

2

*T*2 0.20260 2 2

*S*  *T*1  0.40447 3.97 *E*  *S*4  3.97  0.9925

4

4

*T*4 0.10180 4 4

*S*  *T*1  0.40447  4.23 *E*  *S*5  4.23  0.846

5

5

*T*5 0.09562 5 5

*S*  *T*1  0.40447  8.38 *E*  *S*10  8.38  0.838

10 0.04828 10 10 10

*T*

10

Программа демонстрирует ускорение и эффективность при увеличении числа процессоров . А также увеличение эффективности с ростом кол-ва процессов. Алгоритм не теряет своей скорости и точности с кол-ом процессов