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 4.

# Для вычисления двукратного интеграла f(x,y) = с точностью ε = 106 методом повторного применения квадратурной формулы и равномерной загрузкой всех процессорных элементов, давайте реализуем MPI-программу. Используем метод левых прямоугольников для вычисления интеграла, распараллеливая вычисления по обоим измерениям. Затем оценим ускорение и эффективность программы.

Вот пример реализации на языке C++ с использованием MPI:

#include <iostream>

#include <mpi.h>

#include <math.h>

#include <iomanip>

using namespace std;

double f(double x, double y) {

return x\*pow((x+y),2)/(1+x\*y);

}

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

double ans = 37.4617632175;

int rank, size, rc;

double I = 0;

const int n = 10000;

const double a1 = 1, a2 = 0, b1 = 4, b2 = 1;

MPI\_Comm comm;

rc = MPI\_Init(&argc, &argv);

comm = MPI\_COMM\_WORLD;

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

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

double local\_sum = 0;

const double hx = (b1 - a1) / n;

const double hy = (b2 - a2) / n;

double t = MPI\_Wtime();

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

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

double x = a1 + (i+0.5) \* hx;

double y = a2 + (j+0.5) \* hy;

local\_sum += f(x, y) \* hy\*hx;

}

}

rc = MPI\_Reduce(&local\_sum, &I, 1, MPI\_DOUBLE, MPI\_SUM, 0, comm);

if (rank == 0) {

cout << "Amount of ranks: "<<size<<endl;

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

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

cout << "Time: " << MPI\_Wtime() - t << endl;

}

rc = MPI\_Finalize();

return 0;

}

После выполнения программы на 2 процессах был получен результат:

Result: 37.46176309713

Error: 1.203676092132e-07

Time: 0.2921500205994

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

Приведем результаты расчета программы для *n*  10000:

Количество процессоров size = 1

integral = 37.46176309713 time = 0.5836410522461

Количество процессоров size = 2

integral = 37.46176309713 time = 0.2921500205994

Количество процессоров size = 4

integral = 37.46176309713 time = 0 0.1464478969574

Количество процессоров size = 5

integral = 37.46176309713 time = 0.1176700592041

Количество процессоров size = 10

integral = 37.46176309713 time = 0.05955195426941

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение

*SP*  *T*1 / *Tp* и эффективность

*EP*  *Sp* / *p* :

*S*  *T*1  0.58364  2. 0 *E*  *S*2  2  1

2

2

*T*2 0.29215 2 2

*S*  *T*1  0.58364  3.99 *E*  *S*4  3.99  0.9975

4

4

*T*4 0.14644 4 4

*S*  *T*1  0.58364  4.96 *E*  *S*5  4.96  0.992

5

5

*T*5 0.11767 5 5

*S*  *T*1  0.58364  9.8 *E*  *S*10  9.8  0.98

10 0.05955 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