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 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 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 pow(x, 2) \* pow(sin(x + y) + 1.5, 2);

}

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

double ans = 18.2733345388;

int rank, size, rc;

double I = 0;

const int n = 10000;

const double a1 = -1, a2 = 1, b1 = 4, b2 = 2;

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;

}

}

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

if (rank == 0) {

I \*= hx;

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 1 процессе был получен результат:

Result: 18.27333421865

Error: 3.201540152986e-07

Time: 1.129092931747

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*  10000:

Количество процессоров size = 1

integral = 18.27333421865 time = 1.12909

Количество процессоров size = 2

integral = 18.27333421865 time = 0.56489

Количество процессоров size = 4

integral = 18.27333421865 time = 0 0.30016

Количество процессоров size = 5

integral = 18.27333421865 time = 0.40324

Количество процессоров size = 10

integral = 18.27333421865 time = 0.12924

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение

*SP*  *T*1 / *Tp* и эффективность

*EP*  *Sp* / *p* :

*S*  *T*1  1.12909  1.99 *E*  *S*2  1.99  0.99

2

2

*T*2 0.56489 2 2

*S*  *T*1  1.12909  3.76 *E*  *S*4  3.76  0.94

4

4

*T*4 0.30016 4 4

*S*  *T*1  1.12909  2.8 *E*  *S*5  2.8  0.56

5

5

*T*5 0.40324 5 5

*S*  *T*1  1.12909  8.73 *E*  *S*10  0.38  0.874

10 0.12924 10 10 10

*T*

10

Программа демонстрирует ускорение и эффективность при увеличении числа процессоров до определенного предела, после чего ускорение уменьшается при 5 процессорах, но потом вновь растёт. Например, при четырёх процессорах ускорение составляет примерно 3.76, что означает, что программа работает примерно в 3.76 раза быстрее, чем при одном процессоре. Однако при увеличении числа процессоров до 5 ускорение снижается до 2.8, а эффективность до 0.56, но при 10 процессорах ускорение уже составляет 8.37, а эффективность 0.874. Это может быть связано с увеличением накладных расходов на управление процессами и коммуникацию между ними при большем числе процессоров.