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) = $\int_1^4 \int_0^1 \frac{x*(x+y)^2}{1+x*y} dy\, dx$ с точностью ε = 10^6 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 = 0.1464478969574

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 $S_P = T_1 / T_p$ и эффективность $E_P = S_p / p$:

$$S_2 = \frac{T_1}{T_2} = \frac{0.58364}{0.29215} = 2.0 \qquad E_2 = \frac{S_2}{2} = \frac{2}{2} = 1$$

$$S_4 = \frac{T_1}{T_4} = \frac{0.58364}{0.14644} = 3.99 \qquad E_4 = \frac{S_4}{4} = \frac{3.99}{4} = 0.9975$$

$$S_5 = \frac{T_1}{T_5} = \frac{0.58364}{0.11767} = 4.96 \qquad E_5 = \frac{S_5}{5} = \frac{4.96}{5} = 0.992$$

$$S_{10} = \frac{T_1}{T_{10}} = \frac{0.58364}{0.05955} = 9.8 \qquad E_{10} = \frac{S_{10}}{10} = \frac{9.8}{10} = 0.98$$

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