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 ( )= $\frac{1.4+\sqrt{x+e^x}}{x+1}$ 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 = 0.1018059253693

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

$$S_2 = \frac{T_1}{T_2} = \frac{0.40447}{0.20260} = 2.0 \qquad E_2 = \frac{S_2}{2} = \frac{2}{2} = 1$$

$$S_4 = \frac{T_1}{T_4} = \frac{0.40447}{0.10180} = 3.97 \qquad E_4 = \frac{S_4}{4} = \frac{3.97}{4} = 0.9925$$

$$S_5 = \frac{T_1}{T_5} = \frac{0.40447}{0.09562} = 4.23 \qquad E_5 = \frac{S_5}{5} = \frac{4.23}{5} = 0.846$$

$$S_{10} = \frac{T_1}{T_{10}} = \frac{0.40447}{0.04828} = 8.38 \qquad E_{10} = \frac{S_{10}}{10} = \frac{8.38}{10} = 0.838$$

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