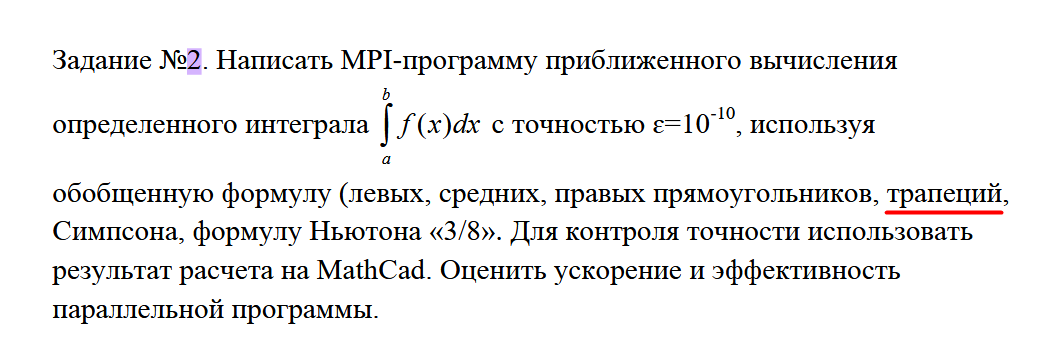
Контрольная №4

Задача на приближенное вычисление определенного интеграла

Боязитов Вадим, 932125

01.05.2025

## Задание 1



### Алгоритм

1.Инициализация MPI

Запускаем MPI среду, определяем ранг процесса и общее количество процессов.

2. Определение функции

Функция calculate\_function(x) вычисляет значение подынтегральной функции arctan(0.7\*x)/(x+1.48).

3. Разделение интервала

Весь интервал интегрирования [0.2, 0.5] делится на равные части по количеству процессов.

Каждый процесс получает свой локальный сегмент [local\_a, local\_b].

4. Метод трапеций:

Функция integrate\_segment вычисляет интеграл на локальном сегменте методом трапеций:

a. Разбивает сегмент на n частей

b. Вычисляет сумму значений функции на границах и в точках разбиения

c. Умножает сумму на шаг интегрирования

5. Сбор результатов

С помощью MPI\_Reduce результаты всех процессов суммируются на процессе 0

### Программная реализация

Окружение Linix. Язык C.

[https://github.com/katet3/tsu\_parallel\_programming/blob/main/lab\_4/lab\_4.c](https://github.com/katet3/tsu_parallel_programming/tree/main/lab_3)

/\*

https://www.wolframalpha.com/input?i=integrate+arctan%280.7\*x%29%2F%28x%2B1.48%29+from+0.2+to+0.5

http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D1%8B\_%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B8%D0%BA%D0%BE%D0%B2\_%D0%B8\_%D1%82%D1%80%D0%B0%D0%BF%D0%B5%D1%86%D0%B8%D0%B9

\*/

/\*

Ускорение S(p) = T(1) / T(p)

T(1) - время выполнения последовательной версии

T(p) - время выполнения параллельной версии на p процессах

\*/

/\*

Эффективность E(p) = S(p) / p => T(1) / (p \* T(p))

\*/

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <mpi.h>

#define INTERVAL\_START 0.2

#define INTERVAL\_END 0.5

#define FIXED\_N 1000000 // Фиксированное количество разбиений

double calculate\_function(double x) {

return atan(0.7 \* x) / (x + 1.48);

}

double integrate\_segment(double a, double b, int n) {

double h = (b - a) / n;

double sum = 0.5 \* (calculate\_function(a) + calculate\_function(b));

for (int i = 1; i < n; i++) {

sum += calculate\_function(a + i \* h);

}

return h \* sum;

}

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

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

double start\_time = MPI\_Wtime();

// Распределение работы между процессами

double segment\_size = (INTERVAL\_END - INTERVAL\_START) / size;

double local\_a = INTERVAL\_START + rank \* segment\_size;

double local\_b = INTERVAL\_START + (rank + 1) \* segment\_size;

// Вычисление интеграла с фиксированным n

int local\_n = FIXED\_N / size; // Распределяем общее n по процессам

double local\_result = integrate\_segment(local\_a, local\_b, local\_n);

// Сбор результатов со всех процессов

double global\_result;

MPI\_Reduce(&local\_result, &global\_result, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

double end\_time = MPI\_Wtime();

double max\_time;

double local\_time = end\_time - start\_time;

MPI\_Reduce(&local\_time, &max\_time, 1, MPI\_DOUBLE, MPI\_MAX, 0, MPI\_COMM\_WORLD);

if (rank == 0) {

// Выполняем последовательную версию для сравнения

double serial\_start = MPI\_Wtime();

double serial\_result = integrate\_segment(INTERVAL\_START, INTERVAL\_END, FIXED\_N);

double serial\_time = MPI\_Wtime() - serial\_start;

printf("Integral result: %.15f\n", global\_result);

printf("Fixed n per process: %d\n", local\_n);

printf("Total n: %d\n", FIXED\_N);

printf("Parallel time: %.6f seconds\n", max\_time);

printf("Number of processes: %d\n", size);

printf("\nSerial result: %.15f\n", serial\_result);

printf("Serial time: %.6f seconds\n", serial\_time);

// Рассчитываем ускорение и эффективность

double speedup = serial\_time / max\_time;

double efficiency = speedup / size \* 100.0;

printf("\nSpeedup: %.2f\n", speedup);

printf("Efficiency: %.2f%%\n", efficiency);

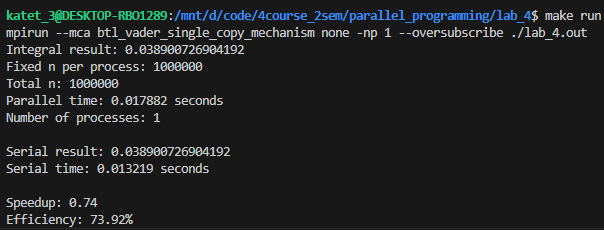
}

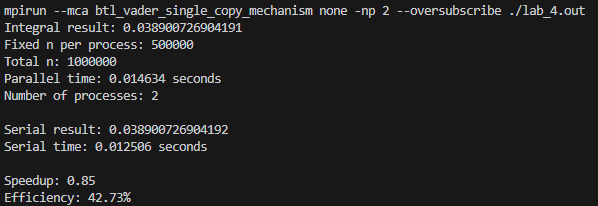
MPI\_Finalize();

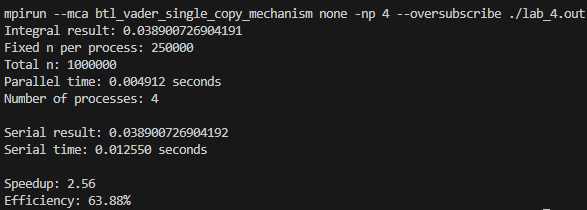
return 0;

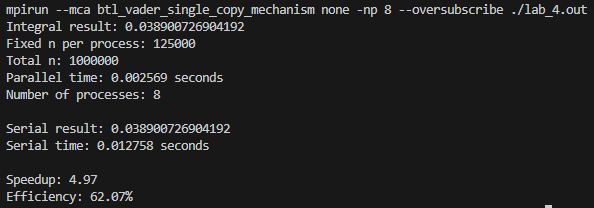
}

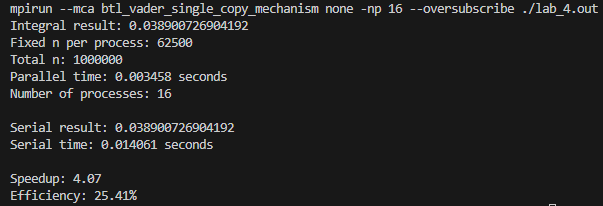
### Тестирование

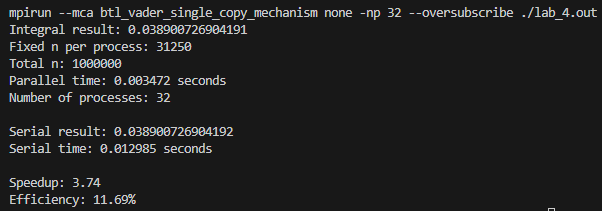


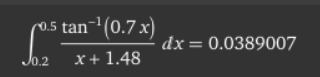












### Вывод

### С ростом кол-ва процессов, уменьшается время на вычисление.Ускорение увеличивается, а эффективность будет падать за счет большого кол-ва процессов.