

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

**ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ.
WORK10**

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНиИТ
Вильцева Данила Денисовича

Проверил

Старший преподаватель

М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

1	Work 10.....	3
1.1	Условие задачи.....	3
1.2	Решение. Последовательная версия метода прямоугольников	3
1.3	Решение. Параллельная версия метода прямоугольников	5
1.4	Решение. Последовательная версия метода Симпсона	7
1.5	Решение. Параллельная версия метода Симпсона	9
2	Результат работы	11
3	Характеристики компьютера.....	13

1 Work 10

1.1 Условие задачи

Аналогично работе с OMP выполните следующее задание через MPI.

Реализуйте параллельные алгоритмы, использующие метод прямоугольников и формулу Симпсона для подсчета интегралов. Точные значения интегралов указаны для проверки численных вычислений. В случае, если в верхнем пределе интегрирования указан знак бесконечности, то в расчете необходимо заменить его на 10 в 6 степени.

Вариант №1

$$1. \int_0^1 \frac{dx}{\sqrt{1-x^2}} = \frac{\pi}{2}$$

1.2 Решение. Последовательная версия метода прямоугольников

```
#include <iostream>
#include <mpi.h>
#include <time.h>
#include <cmath>

using namespace std;

double func(double x)
{
    return (1.0 / sqrt(1.0 - x * x));
}

double integral(const double a, const double b, const double h) {
    int i, n;
    double sum, res = 0; // локальная переменная для подсчета интеграла
    double x; // координата точки сетки
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    for (i = 0; i < n; i++)
    {
        x = a + i * h + h / 2.0;
        sum += func(x) * h;
    }
}
```

```

        res = sum;

        return res;
    }

void integralSimpson(const double a, const double b,
    const double h, double* res) {
    int k, n;
    double sum; // локальная переменная для подсчета интеграла
    double x1, x2; // координата точки сетки
    double f1 = 0.0, f2 = 0.0;
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    for (k = 1; k <= n; k++) {
        double x1 = a + (2 * k - 1) * h;
        double x2 = a + 2 * k * h;
        f1 += func(x1);
        if (k < n)
            f2 += func(x2);
    }
    sum = (h / 3.0) * (func(a) + func(b) + 4 * f1 + 2 * f2);

    *res = sum;
}

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

    double stime, ftime, time, res; // время начала и конца расчета
    double a = 0.0; // левая граница интегрирования
    double b = 1.0; // правая граница интегрирования
    double h = 0.001; // шаг интегрирования
    int n = 6;
    stime = clock();
    res = integral(a, b, h); // вызов функции интегрирования
    ftime = clock();
    time = (ftime - stime) / CLOCKS_PER_SEC;

    // вывод результатов эксперимента
    cout << "execution time : " << time << "\n";
}

```

```

    cout << "integral value : " << res << endl;
    return 0;
}

```

1.3 Решение. Параллельная версия метода прямоугольников

```

#include <iostream>
#include <mpi.h>
#include <time.h>
#include <cmath>

using namespace std;

double func(double x)
{
    return (1.0 / sqrt(1.0 - x * x));
}

double integral(const double a, const double b, const double h) {
    int i, n;
    double sum, res = 0; // локальная переменная для подсчета интеграла
    double x; // координата точки сетки
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    int commsize; // *
    int rank; // *
    double Result; // *

    MPI_Init(NULL, NULL); // *
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // *
    MPI_Comm_size(MPI_COMM_WORLD, &commsize); // *
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); // *

    for (i = 0; i < n; i++)
    {
        x = a + i * h + h / 2.0;
        sum += func(x) * h;
    }

    /*res = sum;*/

    MPI_Reduce(&sum, &Result, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD); // *
    MPI_Finalize(); // *
}

```

```

        return Result;
    }

void integralSimpson(const double a, const double b,
    const double h, double* res) {
    int k, n;
    double sum; // локальная переменная для подсчета интеграла
    double x1, x2; // координата точки сетки
    double f1 = 0.0, f2 = 0.0;
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    for (k = 1; k <= n; k++) {
        double x1 = a + (2 * k - 1) * h;
        double x2 = a + 2 * k * h;
        f1 += func(x1);
        if (k < n)
            f2 += func(x2);
    }
    sum = (h / 3.0) * (func(a) + func(b) + 4 * f1 + 2 * f2);

    *res = sum;
}

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

    double stime, ftime, time, res; // время начала и конца расчета
    double a = 0.0; // левая граница интегрирования
    double b = 1.0; // правая граница интегрирования
    double h = 0.001; // шаг интегрирования
    int n = 6;
    stime = clock();
    res = integral(a, b, h); // вызов функции интегрирования
    ftime = clock();
    time = (ftime - stime) / CLOCKS_PER_SEC;

    // вывод результатов эксперимента
    cout << "execution time : " << time << "\n";
    cout << "integral value : " << res << endl;
}

```

```

    return 0;
}

```

1.4 Решение. Последовательная версия метода Симпсона

```

using namespace std;

double func(double x)
{
    return (1.0 / sqrt(1.0 - x * x));
}

double integral(const double a, const double b, const double h) {
    int i, n;
    double sum, res = 0; // локальная переменная для подсчета интеграла
    double x; // координата точки сетки
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    int commsize; // *
    int rank; // *
    double Result; // *

    MPI_Init(NULL, NULL); // *
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // *
    MPI_Comm_size(MPI_COMM_WORLD, &commsize); // *
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); // *

    for (i = 0; i < n; i++)
    {
        x = a + i * h + h / 2.0;
        sum += func(x) * h;
    }

    /*res = sum;*/

    MPI_Reduce(&sum, &Result, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD); // *
    MPI_Finalize(); // *

    return Result;
}

double integralSimpson(const double a, const double b,

```

```

const double h) {
    int k, n;
    double sum; // локальная переменная для подсчета интеграла
    double x1, x2; // координата точки сетки
    double f1 = 0.0, f2 = 0.0;
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    for (k = 1; k <= n; k++) {
        double x1 = a + (2 * k - 1) * h;
        double x2 = a + 2 * k * h;
        f1 += func(x1);
        if (k < n)
            f2 += func(x2);
    }

    sum = (h / 3.0) * (func(a) + func(b) + 4 * f1 + 2 * f2);

    return sum;
}

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

    double stime, ftime, time, res; // время начала и конца расчета
    double a = 0.0; // левая граница интегрирования
    double b = 0.9999; // правая граница интегрирования
    double h = 0.001; // шаг интегрирования
    int n = 6;
    stime = clock();
    res = integralSimpson(a, b, h); // вызов функции интегрирования
    ftime = clock();
    time = (ftime - stime) / CLOCKS_PER_SEC;

    // вывод результатов эксперимента
    cout << "execution time : " << time << "\n";
    return 0;
}

```


1.5 Решение. Параллельная версия метода Симпсона

```
double func(double x)
{
    return (1.0 / sqrt(1.0 - x * x));
}

double integral(const double a, const double b, const double h) {
    int i, n;
    double sum, res = 0; // локальная переменная для подсчета интеграла
    double x; // координата точки сетки
    n = (int)((b - a) / h); // количество точек сетки интегрирования

    sum = 0.0;

    int commsize; // *
    int rank; // *
    double Result; // *

    MPI_Init(NULL, NULL); // *
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); // *
    MPI_Comm_size(MPI_COMM_WORLD, &commsize); // *
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); // *

    for (i = 0; i < n; i++)
    {
        x = a + i * h + h / 2.0;
        sum += func(x) * h;
    }

    /*res = sum;*/

    MPI_Reduce(&sum, &Result, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD); // *
    MPI_Finalize(); // *

    return Result;
}

double integralSimpson(const double a, const double b,
    const double h) {
    int k, n;
    double sum; // локальная переменная для подсчета интеграла
    double x1, x2; // координата точки сетки
    double f1 = 0.0, f2 = 0.0;
    n = (int)((b - a) / h); // количество точек сетки интегрирования
```

```

sum = 0.0;

int commsize; // *
int rank; // *
double Result1, Result2; // *

MPI_Init(NULL, NULL); // *
MPI_Comm_rank(MPI_COMM_WORLD, &rank); // *
MPI_Comm_size(MPI_COMM_WORLD, &commsize); // *
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD); // *

for (k = 1; k <= n; k++) {
    double x1 = a + (2 * k - 1) * h;
    double x2 = a + 2 * k * h;
    f1 += func(x1);
    if (k < n)
        f2 += func(x2);
}

MPI_Reduce(&f1, &Result1, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
MPI_Reduce(&f2, &Result2, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

MPI_Finalize();

sum = (h / 3.0) * (func(a) + func(b) + 4 * f1 + 2 * f2);

return sum;
}

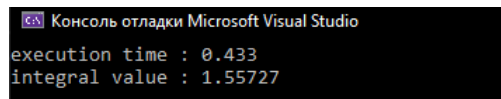
int main(int* argc, char** argv)
{
    double stime, ftime, time, res; // время начала и конца расчета
    double a = 0.0; // левая граница интегрирования
    double b = 0.9999; // правая граница интегрирования
    double h = 0.001; // шаг интегрирования
    int n = 6;
    stime = clock();
    res = integralSimpson(a, b, h); // вызов функции интегрирования
    ftime = clock();
    time = (ftime - stime) / CLOCKS_PER_SEC;

    // вывод результатов эксперимента
    cout << "execution time : " << time << "\n";
    return 0;
}

```

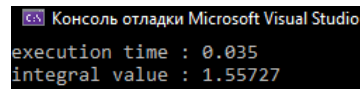
2 Результат работы

Программа, распараллеленная с помощью MPI работает значительно быстрее.



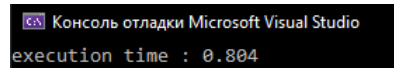
```
Консоль отладки Microsoft Visual Studio
execution time : 0.433
integral value : 1.55727
```

Рисунок 1 – Результат последовательного метода прямоугольников



```
Консоль отладки Microsoft Visual Studio
execution time : 0.035
integral value : 1.55727
```

Рисунок 2 – Результат параллельного метода прямоугольников



```
Консоль отладки Microsoft Visual Studio
execution time : 0.804
```

Рисунок 3 – Результат последовательного метода Симпсона

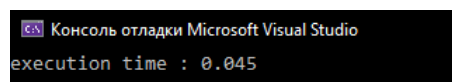


Рисунок 4 – Результат параллельного метода Симпсона

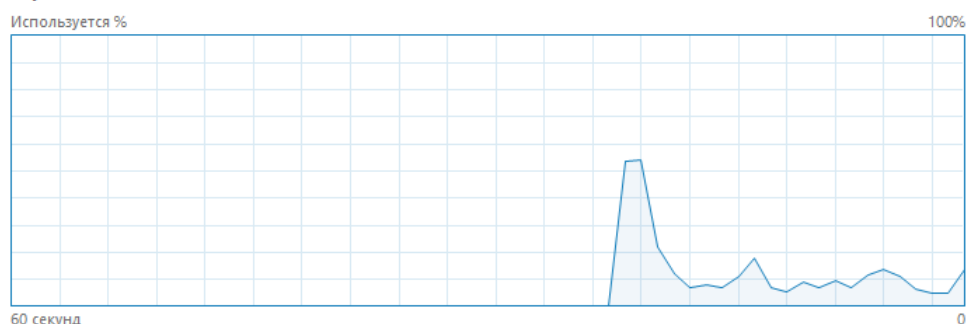
3 Характеристики компьютера

Характеристики устройства

Имя устройства	DESKTOP-MSS8D39
Процессор	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 3.20 GHz
Оперативная память	8,00 ГБ
Код устройства	E3BB953D-13B0-42A7-944B-1ED9FD0E C328
Код продукта	00330-80000-00000-AA153
Тип системы	64-разрядная операционная система, процессор x64

ЦП

Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz



Использование	Скорость	Базовая скорости:	3,20 ГГц
14%	3,43 ГГц	Сокетов:	1
Процессы	Потоки	Ядра:	4
220	3285	Логических процессоров:	4
Время работы	Дескрипторы	Виртуализация:	Включено
100:23:51:24	170005	Кэш L1:	256 КБ
		Кэш L2:	1,0 МБ
		Кэш L3:	6,0 МБ