Задание

Написать MPI-программу вычисления определенного интеграла, используя обобщенную квадратурную формулу Ньютона(«3/8»):

$$\int_a^b f(x)dx = \frac{b-a}{8n}\left[f(a) + f(b) + \sum_{i=1}^{3n-1}\begin{Bmatrix} 2 * f(a+i*h) \text{ если i кратно 3} \\ 3 * f(a+i*h) \text{ если i не кратно 3} \end{Bmatrix}\right]$$

$$h = \frac{b-a}{2*n}$$

Обеспечить равномерную загрузку всех процессорных элементов, участвующих в работе программы. Вычислить ускорение и эффективность программы.

Лабораторная работа №4

Интеграл

$$\int_{2.5}^{5} \frac{e^{-tan(0.8*x)}}{1.35 + cos(x)}dx = 9.287726784951$$

Листинг программы

```
#include <stdio.h>
#include <mpi.h>
#include <math.h>

double f(double x) {
    return exp(-tan(0.8 * x)) / (1.35 + cos(x));
}

double integrate(double a, double b, int n) {
    double h = (b - a) / (3 * n);
    double sum = 0.0;
    for (int i = 1; i <= ((3 * n) - 1); i++) {
        double x = a + i * h;
        if (i % 3 == 0) {sum += 2 * f(x);}
        else {sum += 3 * f(x);}
    }
    return ((b - a)/(8 * n))*(f(a) + f(b) + sum);
}
```

```
int main(int argc, char** argv) {
    int rank, size;
    double a = 2.5, b = 5;
    int n = 800;
    double result, local_result;
    double start_time, end_time;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int local_n = n / size;

        MPI_Barrier(MPI_COMM_WORLD);
    start_time = MPI_Wtime();

    local_result = integrate(
    a + (rank * (b - a) / size),
    a + ((rank + 1) * (b - a) / size),
    local_n);

    end_time = MPI_Wtime();
        MPI_Barrier(MPI_COMM_WORLD);

    MPI_Reduce(&local_result, &result, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);

    if (rank == 0) {
            printf("Size: %d \n", size);
        printf("Result: %.10f \n", result);
        printf("Time: %f seconds\n", end_time - start_time);
    }

    MPI_Finalize();

    return 0;
}
```

Описание программы

В коде программы заданы две функции:

- **f(double x)**
- **integrate(double a, double b, int n)**

f(x) вычисляет значение функции в точке **x**.

Функция **integrate(a, b, n)** - вычисляет значение интеграла с помощью обобщенной квадратурной формулы Ньютона(3/8).

Функция **integrate** вызывается на каждом из процессов, где каждому процессу задается свой интервал по формуле:

$$a = a + \frac{rank * (b - a)}{size}$$

$$b = a + \frac{(rank + 1) * (b - a)}{size}$$

Благодаря чему каждый процессор получит свой интервал, следовательно будет выполняться условие равномерно распределенной нагрузки.

Также число n вычисляется в зависимости от количества процессов

$$local\_n = \frac{n}{size}$$

Тестирование программы

Программа была запущена на 1, 2, 4, 8, 16 процессах, результат выполнения представлен ниже:

```
Size: 1
Result: 9.2877267849
Time: 0.000363 seconds

Size: 2
Result: 9.2877267849
Time: 0.000255 seconds

Size: 4
Result: 9.2877267849
Time: 0.000143 seconds

Size: 8
Result: 9.2877267849
Time: 0.000117 seconds

Size: 16
Result: 9.2877267849
Time: 0.000057 seconds
```

Производительность программы

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение: $S_p = \dfrac{T_1}{T_p}$ и эффективность: $E_p = \dfrac{S_p}{p}$

$$S_2 = \frac{T_1}{T_2} = \frac{0.000363}{0.000255} = 1.423 \qquad E_2 = \frac{S_2}{2} = \frac{1.423}{2} = 0.711$$

$$S_4 = \frac{T_1}{T_4} = \frac{0.000363}{0.000143} = 2.358 \qquad E_4 = \frac{S_4}{4} = \frac{2.358}{4} = 0.634$$

$$S_8 = \frac{T_1}{T_8} = \frac{0.000363}{0.000117} = 3.102 \qquad E_8 = \frac{S_8}{8} = \frac{3.102}{8} = 0.387$$

$$S_{16} = \frac{T_1}{T_{16}} = \frac{0.000363}{0.000057} = 6.368 \qquad E_{16} = \frac{S_{16}}{16} = \frac{6.368}{16} = 0.398$$

Хорошее ускорение наблюдается при любом числе параллельных процессов и эффективность параллельного алгоритма очень высокая.

Лабораторная работа №5

В лабораторной работе №5 требуется вычислить двойной интеграл тем же методом, который был описан в заголовке задания.

Интеграл

$$\int_{-1}^{4}\left[\int_{0}^{3}\frac{\sqrt{x^2+2y^2+2.04}}{\sqrt{1+y^2}}dy\right]dx = 28.0515661951$$

Необходимые изменения в программу

Для начала переопределим подинтегральную функцию:

```
double f(double x, double y) {
    return sqrt(x*x + 2*y*y + 2.04) / sqrt(1 + y*y);
}
```

Также переопределим функцию integrate, чтобы она работала с функцией двух переменных, а также добавим параметр y.

```
double integrate(double a, double b, double y, int n) {
    double h = (b - a) / (3 * n);
    double sum = 0.0;
    for (int i = 1; i <= ((3 * n) - 1); i++) {
        double x = a + i * h;
        if (i % 3 == 0) {
            sum += 2 * f(x, y);
        }
        else {
            sum += 3 * f(x, y);
        }
    }
    return ((b - a)/(8 * n))*(f(a, y) + f(b, y) + sum);
}
```

Добавим функцию doubleIntegrate, которая будет вызывать функцию integrate для определенного приближения y:

```
double doubleIntegrate(double a, double b, double c, double d, int n) {
    double result = 0.0;
    double h = (d - c) / (3 * n);
    for (int i = 1; i <= ((3 * n) - 1); i++) {
        double y = c + i * h;
        if (i % 3 == 0) {
            result += 2 * integrate(a, b, y, n);
        }
        else {
            result += 3 * integrate(a, b, y, n);
        }
    }
    return ((d - c)/(8 * n))*(integrate(a, b, c, n) + integrate(a, b,
d, n) + result);
}
```

Внесем небольшие изменения в main функцию. А именно добавим пределы интегрирования

```
....
Код main функции

double a = -1, b = 4;
double c = 0, d = 3;
....
Вызов функции doubleIntegrate
local_result = doubleIntegrate(a + (rank * (b - a) / size),
a + ((rank + 1) * (b - a) / size),
c, d, local_n);
```

Результат выполнения программы

Запустим программу на 1, 2, 4, 8 и 16 процессах

```
Size: 1
Result: 28.0515661951
Time: 0.193212 seconds

Size: 2
Result: 28.0515661951
Time: 0.048505 seconds

Size: 4
Result: 28.0515661951
Time: 0.012128 seconds

Size: 8
Result: 28.0515661951
Time: 0.003052 seconds

Size: 16
Result: 28.0515661951
Time: 0.000760 seconds
```

Производительность программы

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение: $S_p = \frac{T_1}{T_p}$ и эффективность: $E_p = \frac{S_p}{p}$

$$S_2 = \frac{T_1}{T_2} = \frac{0.193212}{0.048505} = 3.983 \qquad E_2 = \frac{S_2}{2} = \frac{3.983}{2} = 1.99$$

$$S_4 = \frac{T_1}{T_4} = \frac{0.193212}{0.012128} = 15.931 \qquad E_4 = \frac{S_4}{4} = \frac{15.931}{4} = 3.982$$

$$S_8 = \frac{T_1}{T_8} = \frac{0.193212}{0.003052} = 63.306 \qquad E_8 = \frac{S_8}{8} = \frac{63.306}{8} = 7.913$$

$$S_{16} = \frac{T_1}{T_{16}} = \frac{0.193212}{0.000760} = 254.226 \qquad E_{16} = \frac{S_{16}}{16} = \frac{254.226}{16} = 15.889$$

Лабораторная работа №6

В лабораторной работе №6 требуется вычислить интеграл тем же методом, что и в лабораторной работе №3, интеграл остается тем же, однако в этом случае требуется использовать библиотеку OpenMP.

Для работы программы перепишем основную функцию с вычислением интеграла:

```
double integrate(double a, double b, int n) {
    double h = (b - a) / (3 * n);
    double sum = 0.0;

#pragma omp parallel for reduction(+:sum)
    for (int i = 1; i <= ((3 * n) - 1); i++) {
        double x = a + i * h;
        if (i % 3 == 0) {
            sum += 2 * f(x);
        }
        else {
            sum += 3 * f(x);
        }
    }

    return ((b - a) / (8 * n)) * (f(a) + f(b) + sum);
}
```

В данном случае **#pragma omp parallel for** - директива, которая указывает компилятору на параллельное выполнение цикла for. Каждая итерация цикла может быть выполнена независимо друг от друга на разных потоках.

Также немного изменим функцию main, в которой происходит вызов функции и замер скорости вычислений.

```
int main(int argc, char** argv) {
    double a = 2.5, b = 5;
    int n = 1000;
    int size = 0;

    double result;
    double start_time, end_time;

    #pragma omp parallel
    {
        start_time = omp_get_wtime();

        result = integrate(a, b, n);

        end_time = omp_get_wtime();

        size = omp_get_max_threads();
    }
    printf("Size: %d \n", size);
    printf("Result: %.10f \n", result);
        printf("Time: %f seconds\n", end_time - start_time);

    return 0;
}
```

Результат выполнения программы

Запустим программу на 1, 2, 4 и 8 процессах;

```
Size: 1
Result: 9.2877267849
Time: 0.025415 seconds

Size: 2
Result: 9.2877267849
Time: 0.022512 seconds

Size: 4
Result: 9.2877267849
Time: 0.018828 seconds

Size: 8
Result: 9.2877267849
Time: 0.016583 seconds
```

Производительность программы

Во всех запусках программа даёт верный результат вычисления интеграла.

Оценим ускорение: $S_p = \frac{T_1}{T_p}$ и эффективность: $E_p = \frac{S_p}{p}$

$$S_2 = \frac{T_1}{T_2} = \frac{0.025415}{0.022512} = 1.129 \qquad E_2 = \frac{S_2}{2} = \frac{1.129}{2} = 0.564$$

$$S_4 = \frac{T_1}{T_4} = \frac{0.025415}{0.018828} = 1.349 \qquad E_4 = \frac{S_4}{4} = \frac{1.349}{4} = 0.337$$

$$S_8 = \frac{T_1}{T_8} = \frac{0.025415}{0.016583} = 1.532 \qquad E_8 = \frac{S_8}{8} = \frac{1.532}{8} = 0.191$$

Заключение

В процессе прохождения курса параллельное программирование, я приобрел навыки работы с библиотеками для реализации параллельных вычислений MPI и OpenMP.
Я изучил особенности разработки программ для каждой из указанных библиотек, а также их функциональные возможности.
В ходе обучения я ознакомился с параллельными алгоритмами, их реализацией и методами оценки эффективности.

В результате завершения курса были выполнены лабораторные работы, включающие в себя реализацию параллельных вычислений определенного и двойного интегралов, передачу данных между процессами, а также операции над матрицами.