

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

**ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ, ФОРМУЛА СИМПСОНА,
ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННЫХ И КРАТНЫХ ИНТЕГРАЛОВ**

ОТЧЕТ О ПРАКТИКЕ

Студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНиИТ
Аношкина Андрея Алексеевича

Проверил

Старший преподаватель

М. С. Портенко

Саратов 2024

СОДЕРЖАНИЕ

1 Work 01 3

2 Work 02..... 8

3 Work 03.....11

1 Work 01

Задание

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 10^6 . Сравните время численного интегрирования для последовательной и параллельной реализации. Какое ускорение выполнения программы предоставляет переход к многопоточной версии?

Вариант задания 2:

$$\int_0^{\infty} \frac{dx}{1+x^2} = \frac{\pi}{2}$$

Метод прямоугольников, формула Симпсона

Метод прямоугольников геометрически заключается в том, что интеграл приближенно представляется в виде суммы площадей элементарных прямоугольников.

Для случая деления отрезка интегрирования на равные части и вычисления функции в центре отрезков:

$$\begin{cases} J = \int_a^b f(x) dx \approx \sum_{i=0}^{N-1} (h \cdot f(x_i)) = h \cdot \sum_{i=0}^{N-1} f(x_i) \\ x_i = a + i \cdot h + \frac{h}{2} \end{cases} \quad (1)$$

где N — количество отрезков интегрирования, а $h = (b - a)/N$

Ступенчатая (stair-case) аппроксимация гладких изогнутых поверхностей, возникает в результате дискретизации модели прямоугольной сеткой. Для борьбы со ступенчатой аппроксимацией может использоваться, например, квадратурная формула Симпсона:

$$\int_a^b f(x) dx \approx \frac{h}{3} [f(a) + f(b) + 4 \sum_{k=1}^N f(a + (2k-1)h) + 2 \sum_{k=1}^{N-1} f(a + 2kh)] \quad (2)$$

где $h = \frac{b-a}{2N}$, $N \gg 1$.

Формула Симпсона геометрически заключается в том, что через три ординаты, отвечающие трем последовательным узлам сетки, проводится парабола

и затем складываются получившиеся при этом площади элементарных криволинейных трапеций.

Реализация

Код решения приведен ниже:

```
1  #include <iostream>
2  #include <omp.h>
3  #include <time.h>
4  #define PI 3.1415926535897932384626433832795
5
6  using namespace std;
7
8  double f1(double x) {
9      return 1.0 / (1 + x * x);
10 }
11
12 void integral_posl(const double a, const double b, const double h, double* res) {
13     double sum = 0;
14     int n = (int)((b - a) / h);
15
16     for (int i = 0; i < n; ++i) {
17         double x = a + i * h + h / 2;
18         sum += f1(x) * h;
19     }
20
21     *res = sum;
22 }
23
24 void integral_parallel(const double a, const double b, const double h, double* res) {
25     double sum = 0;
26     int n = (int)((b - a) / h);
27
28     #pragma omp parallel for reduction(+: sum)
29     for (int i = 0; i < n; ++i) {
30         double x = a + i * h + h / 2;
31         sum += f1(x) * h;
32     }
33
34     *res = sum;
35 }
36
37 void integral_Simpson(const double a, const double b, const double h, double* res) {
38     double sum = f1(a) + f1(b);
39     int n = (int)((b - a) / 2 * h);
40
41     #pragma omp parallel for reduction(+: sum)
42     for (int i = 1; i <= n; ++i) {
43         double x = a + h * (2 * i - 1);
```

```

44         sum += f1(x) * 4;
45     }
46
47     #pragma omp parallel for reduction(+: sum)
48     for (int i = 1; i < n; ++i) {
49         double x = a + 2 * i * h;
50         sum += 2 * f1(x);
51     }
52
53     *res = h / 3 * sum;
54 }
55
56 double experiment(double* res, void f(const double a, const double b, const double h, double* res)) {
57     double stime = 0, ftime = 0;
58     double a = 0;
59     double b = 1e+6;
60     double h = 0.01;
61     stime = clock();
62     f(a, b, h, res);
63     ftime = clock();
64
65     return (ftime - stime) / CLOCKS_PER_SEC;
66 }
67
68 void calculate(void f(const double a, const double b, const double h, double* res)) {
69     double avg_time = 0;
70     double min_time = 0;
71     double max_time = 0;
72     double res = 0;
73     int numbExp = 10;
74
75     min_time = max_time = experiment(&res, f);
76     avg_time = min_time / numbExp;
77     for (int i = 0; i < numbExp - 1; ++i) {
78         double time = experiment(&res, f);
79         if (time > max_time)
80             max_time = time;
81         if (time < min_time)
82             min_time = time;
83
84         avg_time += time / numbExp;
85     }
86
87     cout << "Execution time: " << avg_time << "; " << min_time << "; " << max_time << "\n";
88     cout.precision(8);
89     cout << "Integral value: " << res << "\n";
90 }
91
92 int main() {
93

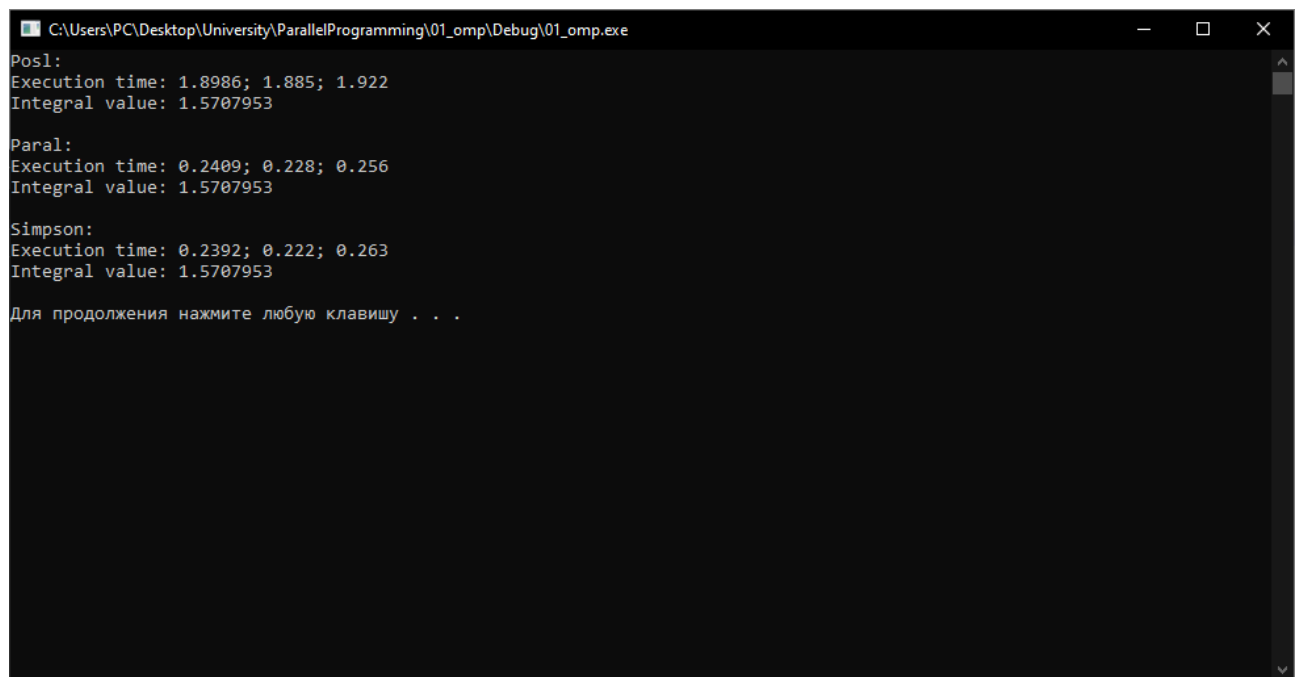
```

```

94     bool calcPosl = true, calcParal = true, calcSimpson = true;
95
96     // Последовательное вычисление
97
98     if (calcPosl) {
99         cout << "Posl:\n";
100        calculate(integral_posl);
101        cout << "\n";
102    }
103
104    // Параллельное вычисление
105
106    if (calcParal) {
107        cout << "Paral:\n";
108        calculate(integral_paral);
109        cout << "\n";
110    }
111
112    // Параллельное вычисление по формуле Симпсона
113
114    if (calcSimpson) {
115        cout << "Simpson:\n";
116        calculate(integral_Simpson);
117        cout << "\n";
118    }
119
120    system("pause");
121    return 0;
122 }

```

Результаты работы



```
C:\Users\PC\Desktop\University\ParallelProgramming\01_omp\Debug\01_omp.exe
Posl:
Execution time: 1.8986; 1.885; 1.922
Integral value: 1.5707953

Paral:
Execution time: 0.2409; 0.228; 0.256
Integral value: 1.5707953

Simpson:
Execution time: 0.2392; 0.222; 0.263
Integral value: 1.5707953

Для продолжения нажмите любую клавишу . . .
```

Рисунок 1 – Work-1

Значение ускорения выполнения программы при переходе к многопоточной версии:

$$a = \frac{timeSeq}{timePar} = \frac{1.8986}{0.2409} \approx 7.8812$$

2 Work 02

Задание

Модифицируйте разработанную ранее программу по методу прямоугольников для численного интегрирования тестовой функции:

$$\int_0^{16} \int_0^{16} \frac{e^{\cos(\pi x) \sin(\pi y)} + 1}{(b_1 - a_1)(b_2 - a_2)} dx dy \approx 2.130997 \quad (3)$$

Предложите несколько способов распараллеливания базового алгоритма метода прямоугольников для двумерной функции с использованием директивы `#pragma omp parallel for`. Реализуйте предложенные способы на примере тестовой функции. Сравните время численного интегрирования для последовательной и параллельных реализаций и параллельных реализаций между собой.

Численное интегрирование по методу прямоугольников для двумерной функции

$$\begin{cases} J = \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x, y) dx dy \approx \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (h_1 \cdot h_2 \cdot f(x_i, y_j)) \\ x_i = a_1 + i \cdot h_1 + \frac{h_1}{2} \\ y_j = a_2 + j \cdot h_2 + \frac{h_2}{2} \end{cases} \quad (4)$$

где N — количество отрезков интегрирования по оси x , M — количество отрезков интегрирования по оси y , $h_1 = (b_1 - a_1)/N$ и $h_2 = (b_2 - a_2)/N$.

Реализация

Фрагмент кода решения приведен ниже:

```
1  #include <iostream>
2  #include <omp.h>
3  #include <time.h>
4  #include <cmath>
5  #define PI 3.1415926535897932384626433832795
6
7  using namespace std;
8
9  double f1(const double x, const double y, const double a1, const double b1, const double a2, const double b2) {
10     return (1 + exp(sin(PI * x) * cos(PI * y))) / (b1 - a1) / (b2 - a2);
11 }
12
```



```

13 void integral_posl(const double a1, const double b1, const double a2, const double b2, const double h1, const
    ↪ double h2, double* res) {
14     double sum = 0;
15     int n = (int)((b1 - a1) / h1);
16     int m = (int)((b2 - a2) / h2);
17
18     for (int i = 0; i < n; ++i) {
19         double x = a1 + i * h1 + h1 / 2;
20         for (int j = 0; j < m; ++j) {
21             double y = a2 + j * h2 + h2 / 2;
22             sum += f1(x, y, a1, b1, a2, b2) * h1 * h2;
23         }
24     }
25
26     *res = sum;
27 }
28
29 void integral_parallel_1(const double a1, const double b1, const double a2, const double b2, const double h1,
    ↪ const double h2, double* res) {
30     double sum = 0;
31     int n = (int)((b1 - a1) / h1);
32     int m = (int)((b2 - a2) / h2);
33
34     #pragma omp parallel for reduction(+: sum)
35     for (int i = 0; i < n; ++i) {
36         double x = a1 + i * h1 + h1 / 2;
37         for (int j = 0; j < m; ++j) {
38             double y = a2 + j * h2 + h2 / 2;
39             sum += f1(x, y, a1, b1, a2, b2) * h1 * h2;
40         }
41     }
42
43     *res = sum;
44 }
45
46 void integral_parallel_2(const double a1, const double b1, const double a2, const double b2, const double h1,
    ↪ const double h2, double* res) {
47     double sum = 0;
48     int n = (int)((b1 - a1) / h1);
49     int m = (int)((b2 - a2) / h2);
50
51     #pragma omp parallel for reduction(+: sum)
52     for (int i = 0; i < n; ++i) {
53         double x = a1 + i * h1 + h1 / 2;
54         #pragma omp parallel for reduction(+: sum)
55         for (int j = 0; j < m; ++j) {
56             double y = a2 + j * h2 + h2 / 2;
57             sum += f1(x, y, a1, b1, a2, b2) * h1 * h2;
58         }
59     }

```

```

60
61     *res = sum;
62 }
63
64 double experiment(double* res, void f(const double a1, const double b1, const double a2, const double b2,
    ↪ const double h1, const double h2, double* res)) {
65     double stime = 0, ftime = 0;
66     double a1 = 0, b1 = 16;
67     double a2 = 0, b2 = 16;
68     double h1 = 0.005, h2 = 0.005;
69     stime = clock();
70     f(a1, b1, a2, b2, h1, h2, res);
71     ftime = clock();
72
73     return (ftime - stime) / CLOCKS_PER_SEC;
74 }
75
76 void calculate(void f(const double a1, const double b1, const double a2, const double b2, const double h1,
    ↪ const double h2, double* res)) {
77     double avg_time = 0;
78     double min_time = 0;
79     double max_time = 0;
80     double res = 0;
81     int numbExp = 10;
82
83     min_time = max_time = experiment(&res, f);
84     avg_time = min_time / numbExp;
85     for (int i = 0; i < numbExp - 1; ++i) {
86         double time = experiment(&res, f);
87         if (time > max_time)
88             max_time = time;
89         if (time < min_time)
90             min_time = time;
91
92         avg_time += time / numbExp;
93     }
94
95     cout << "Execution time: " << avg_time << "; " << min_time << "; " << max_time << "\n";
96     cout.precision(8);
97     cout << "Integral value: " << res << "\n";
98 }
99
100 int main() {
101
102     bool calcPosl = true, calcParal_1 = true, calcParal_2 = true;
103
104     // Последовательное вычисление
105
106     if (calcPosl) {
107         cout << "Posl:\n";

```

```

108         calculate(integral_posl);
109         cout << "\n";
110     }
111
112     // Параллельное вычисление (1)
113
114     if (calcParal_1) {
115         cout << "Paral_1:\n";
116         calculate(integral_paral_1);
117         cout << "\n";
118     }
119
120     // Параллельное вычисление (2)
121
122     if (calcParal_2) {
123         cout << "Paral_2:\n";
124         calculate(integral_paral_2);
125         cout << "\n";
126     }
127
128     system("pause");
129     return 0;
130 }

```

Результаты работы

```

Выбрать C:\Users\PC\Desktop\University\ParallelProgramming\02_omp\Debug\02_omp.exe
Posl:
Execution time: 0.9053; 0.901; 0.922
Integral value: 2.1309969

Paral_1:
Execution time: 0.1054; 0.094; 0.117
Integral value: 2.1309969

Paral_2:
Execution time: 0.1037; 0.09; 0.128
Integral value: 2.1309969

Для продолжения нажмите любую клавишу . . .

```

Рисунок 2 – Work-2

3 Work 03

Задание

Реализовать параллельно один из методов приближенного вычисления двойных интегралов, а именно метод статистических испытаний.

Метод трапеций

Пусть и по направлению x , и по направлению y для приближенного вычисления применяется формула трапеций.

$$F(y_j) \approx h_1 * \sum_{i=0}^m q_{1,i} * f(x_i, y_j)$$

где $q_{1,i} = 0.5$, при $i = 0$ и $i = m$; $q_{1,i} = 1$, при $i = 1, 2, \dots, m - 1$
и

$$J \approx h_2 * \sum_{j=0}^n q_{2,j} * F(y_j)$$

где $q_{2,j} = 0.5$, при $j = 0$ и $j = n$; $q_{2,j} = 1$, при $j = 1, 2, \dots, n - 1$
Тогда

$$J \approx h_1 * h_2 * \sum_{i=0}^m q_{1,i} * \sum_{j=0}^n q_{2,j} * f(x_i, y_j)$$

где $q_{ij} = q_{1,i} * q_{2,j}$

Реализация

Фрагмент кода решения приведен ниже:

```
1  #include <iostream>
2  #include <omp.h>
3  #include <time.h>
4  #include <cmath>
5  #define PI 3.1415926535897932384626433832795
6
7  using namespace std;
8
9  double f1(const double x, const double y, const double a1, const double b1, const double a2, const double b2) {
10     return (1 + exp(sin(PI * x) * cos(PI * y))) / (b1 - a1) / (b2 - a2);
11 }
12
13 double q(int i, int j, int n, int m) {
14     double q = 1;
15     if (i == 0 || i == n)
16         q *= 0.5;
17     if (j == 0 || j == m)
18         q *= 0.5;
19     return q;
20 }
21
22 void integral(const double a1, const double b1, const double a2, const double b2, const double h1, const
↪ double h2, double* res) {
```

```

23     double sum = 0;
24     int n = (int)((b1 - a1) / h1);
25     int m = (int)((b2 - a2) / h2);
26
27     #pragma omp parallel for reduction(+: sum)
28     for (int i = 0; i <= n; ++i) {
29         double x = a1 + i * h1 + h1 / 2;
30         for (int j = 0; j <= m; ++j) {
31             double y = a2 + j * h2 + h2 / 2;
32             sum += f1(x, y, a1, b1, a2, b2) * h1 * h2 * q(i, j, n, m);
33         }
34     }
35
36     *res = sum;
37 }
38
39 double experiment(double* res, void f(const double a1, const double b1, const double a2, const double b2,
↪ const double h1, const double h2, double* res)) {
40     double stime = 0, ftime = 0;
41     double a1 = 0, b1 = 16;
42     double a2 = 0, b2 = 16;
43     double h1 = 0.005, h2 = 0.005;
44     stime = clock();
45     f(a1, b1, a2, b2, h1, h2, res);
46     ftime = clock();
47
48     return (ftime - stime) / CLOCKS_PER_SEC;
49 }
50
51 void calculate(void f(const double a1, const double b1, const double a2, const double b2, const double h1,
↪ const double h2, double* res)) {
52     double avg_time = 0;
53     double min_time = 0;
54     double max_time = 0;
55     double res = 0;
56     int numbExp = 10;
57
58     min_time = max_time = experiment(&res, f);
59     avg_time = min_time / numbExp;
60     for (int i = 0; i < numbExp - 1; ++i) {
61         double time = experiment(&res, f);
62         if (time > max_time)
63             max_time = time;
64         if (time < min_time)
65             min_time = time;
66
67         avg_time += time / numbExp;
68     }
69
70     cout << "Execution time: " << avg_time << "; " << min_time << "; " << max_time << "\n";

```

```
71     cout.precision(8);
72     cout << "Integral value: " << res << "\n";
73 }
74
75 int main() {
76
77     calculate(integral);
78     cout << "\n";
79
80     system("pause");
81     return 0;
82 }
```

Результат работы

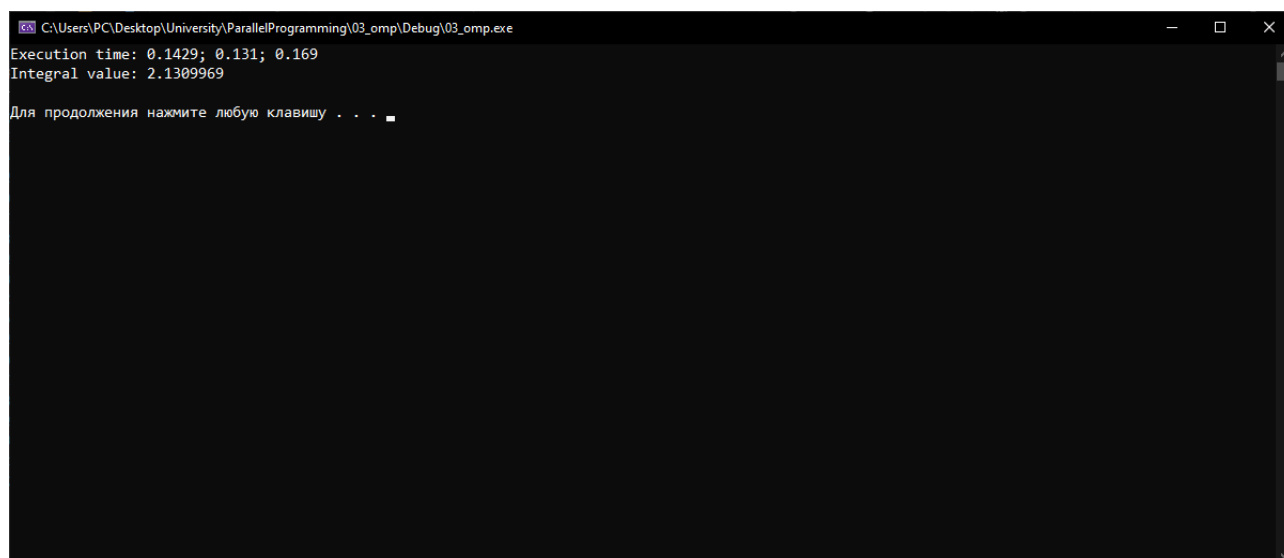


Рисунок 3 – Work-3