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 11 3

1 Work 11

Задание

Аналогично работе с OMP выполните следующее задание через MPI.

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

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

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

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

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:

```
#include <iostream>
#include "mpi.h"
#include <time.h>
#include <cmath>
#define PI 3.14159265358979323846264338327950
using namespace std;
int NProc, ProcId;
MPI_Status st;
```

```
12 double f1(const double x, const double y, const double a1, const double b1, const double a2, const double b2) {
13         return (1 + exp(sin(PI * x) * cos(PI * y))) / (b1 - a1) / (b2 - a2);
14 }
15
16 void integral_posl(const double a1, const double b1, const double a2, const double b2, const double h1, const
   ↪  double h2, double* res) {
17         double sum = 0;
18         int n = (int)((b1 - a1) / h1);
19         int m = (int)((b2 - a2) / h2);
20
21         for (int i = 0; i < n; ++i) {
22                 double x = a1 + i * h1 + h1 / 2;
23                 for (int j = 0; j < m; ++j) {
24                         double y = a2 + j * h2 + h2 / 2;
25                         sum += f1(x, y, a1, b1, a2, b2) * h1 * h2;
26                 }
27         }
28
29         *res = sum;
30 }
31
32 void integral_paral_1(const double a1, const double b1, const double a2, const double b2, const double h1,
   ↪  const double h2, double* res) {
33         double sum = 0;
34         int n = (int)((b1 - a1) / h1);
35         int m = (int)((b2 - a2) / h2);
36
37         for (int i = ProcId; i < n; i += NProc) {
38                 double x = a1 + i * h1 + h1 / 2;
39                 for (int j = 0; j < m; ++j) {
40                         double y = a2 + j * h2 + h2 / 2;
41                         sum += f1(x, y, a1, b1, a2, b2) * h1 * h2;
42                 }
43         }
44
45         if (ProcId != 0)
46                 MPI_Send(&sum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
47         else {
48                 for (int i = 1; i < NProc; ++i) {
49                         double s;
50                         MPI_Recv(&s, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
                           ↪  &st);
51                         sum += s;
52                 }
53
54                 *res = sum;
55         }
56 }
57
```

```
58 void integral_paral_2(const double a1, const double b1, const double a2, const double b2, const double h1,
   ↪ const double h2, double* res) {
59     double sum = 0;
60     int n = (int)((b1 - a1) / h1);
61     int m = (int)((b2 - a2) / h2);
62 
63     for (int i = ProcId / 2; i < n; i += NProc / 2) {
64         double x = a1 + i * h1 + h1 / 2;
65         for (int j = ProcId % 2; j < m; j += NProc / 2) {
66             double y = a2 + j * h2 + h2 / 2;
67             sum += f1(x, y, a1, b1, a2, b2) * h1 * h2;
68         }
69     }
70 
71     if (ProcId != 0)
72         MPI_Send(&sum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
73     else {
74         for (int i = 1; i < NProc; ++i) {
75             double s;
76             MPI_Recv(&s, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
                ↪ &st);
77             sum += s;
78         }
79 
80         *res = sum;
81     }
82 }
83 
84 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)) {
85     double stime = 0, ftime = 0;
86     double a1 = 0, b1 = 16;
87     double a2 = 0, b2 = 16;
88     double h1 = 0.005, h2 = 0.005;
89     stime = clock();
90     f(a1, b1, a2, b2, h1, h2, res);
91     ftime = clock();
92 
93     return (ftime - stime) / CLOCKS_PER_SEC;
94 }
95 
96 void calculate(void f(const double a1, const double b1, const double a2, const double b2, const double h1,
   ↪ const double h2, double* res)) {
97     double avg_time = 0;
98     double min_time = 0;
99     double max_time = 0;
100    double res = 0;
101    int numbExp = 10;
102 
103    min_time = max_time = experiment(&res, f);
```

```
104         avg_time = min_time / numbExp;
105         for (int i = 0; i < numbExp - 1; ++i) {
106                 double time = experiment(&res, f);
107                 if (time > max_time)
108                         max_time = time;
109                 if (time < min_time)
110                         min_time = time;
111 
112                 avg_time += time / numbExp;
113         }
114 
115         if (ProcId == 0) {
116                 cout << "Execution time: " << avg_time << "; " << min_time << "; " << max_time <<
                    ↪   "\n";
117                 cout.precision(8);
118                 cout << "Integral value: " << res << "\n";
119         }
120 }
121 
122 int main() {
123 
124         bool calcPosl = true, calcParal_1 = true, calcParal_2 = true;
125 
126         MPI_Init(NULL, NULL);
127         MPI_Comm_size(MPI_COMM_WORLD, &NProc);
128         MPI_Comm_rank(MPI_COMM_WORLD, &ProcId);
129 
130         // Последовательное вычисление
131 
132         if (calcPosl && ProcId == 0) {
133                 cout << "Posl:\n";
134                 calculate(integral_posl);
135                 cout << "\n";
136         }
137 
138         // Параллельное вычисление (1)
139 
140         if (calcParal_1) {
141                 if (ProcId == 0)
142                         cout << "Paral_1:\n";
143                 calculate(integral_paral_1);
144                 if (ProcId == 0)
145                         cout << "\n";
146         }
147 
148         // Параллельное вычисление (2)
149 
150         if (calcParal_2) {
151                 if (ProcId == 0)
152                         cout << "Paral_2:\n";
```

```
153             calculate(integral_paral_2);
154             if (ProcId == 0)
155                 cout << "\n";
156         }
157
158         MPI_Finalize();
159
160         return 0;
161     }
```

Результаты работы

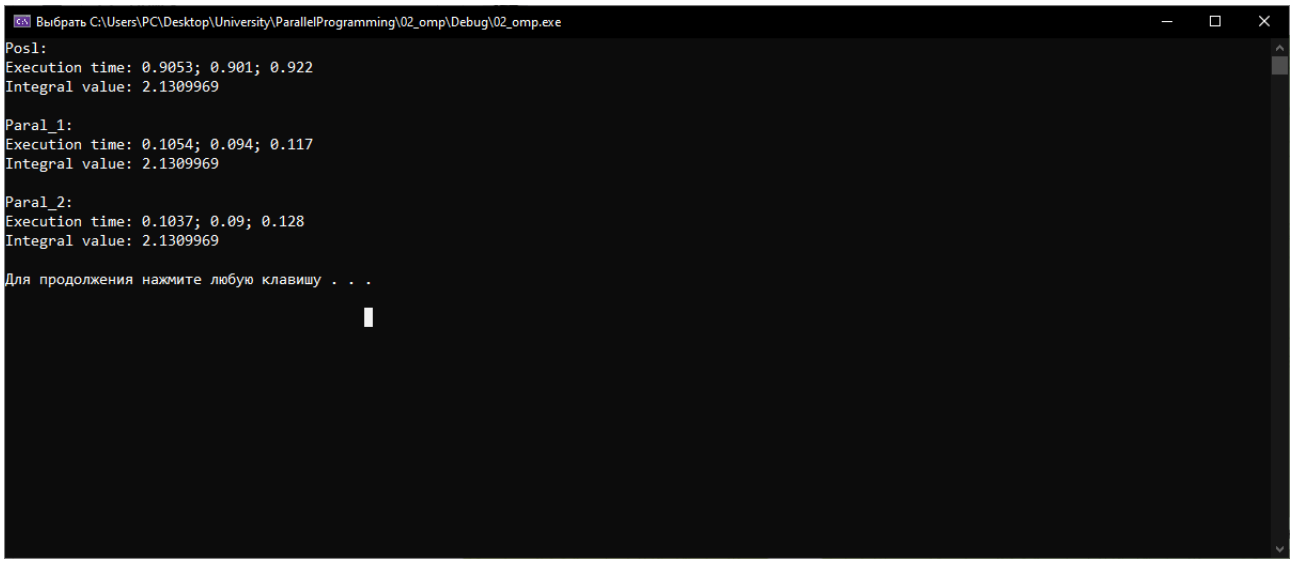

Рисунок 1 – Work-2 (OpenMP)

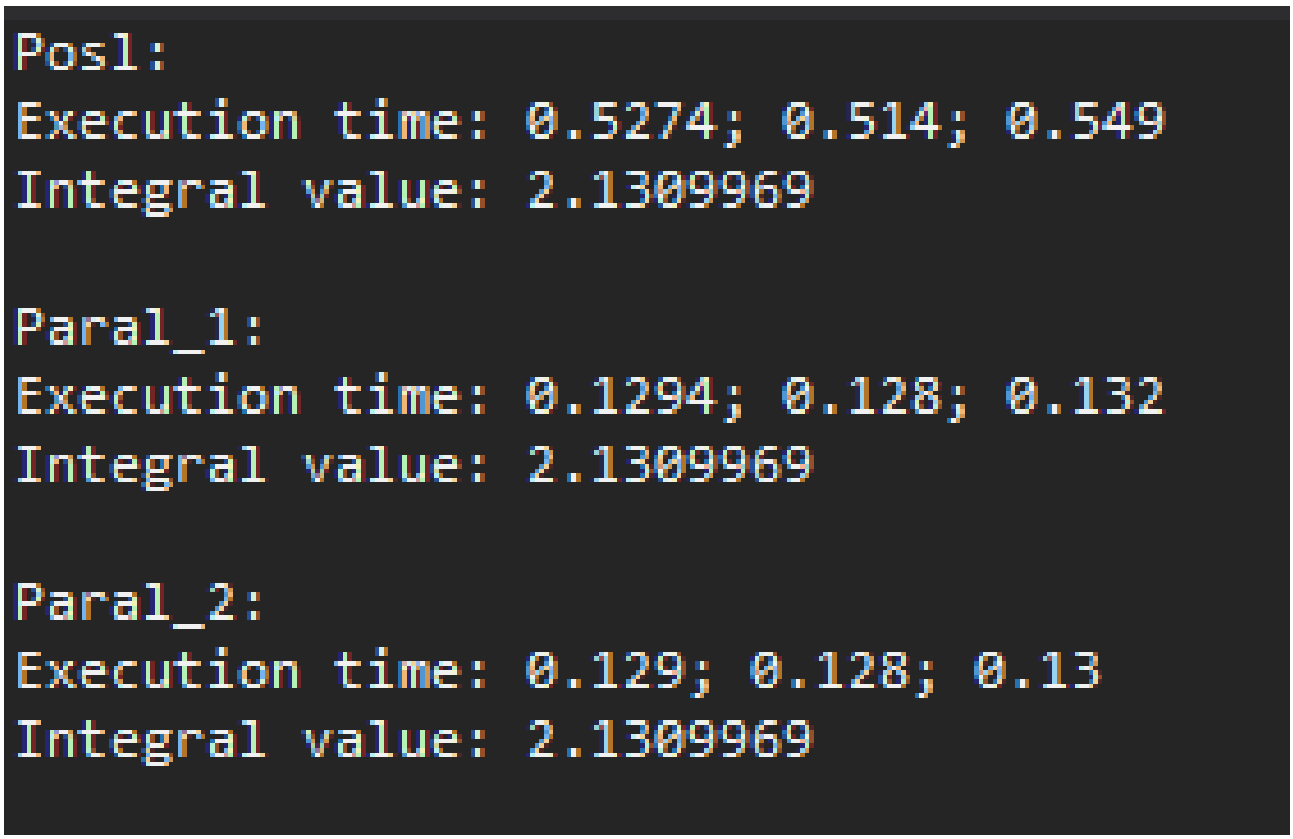

Рисунок 2 – Work-11 (OpenMPI)

Статическое распределение задач по потокам показало наилучший результат. Динамическое распределение задач проиграло по времени в связи с необходимостью обмениваться сообщениями между потоками и диспетчером, но в случае более тяжелых задач является более эффективным.

Характеристики устройства

Процессор: Intel(R) Core(TM) i5-10400F

Ядер: 6

Оперативная память: 16 Гб