**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

Практические приемы построения многопоточных приложений. Вариант 30

**Исполнитель**

Студент группы БПИ195

Чекрий П. С.

Задание

Задача для агронома. Председатель дачного кооператива Сидоров В.И. получил указание, что в связи с составлением единого земельного кадастра, необходимо представить справку о площади занимаемых земель. Известно, что территория с запада и востока параллельна меридианам, на севере ограничены параллелью, а с юга выходят к реке, описываемой функцией f(x). Требуется создать многопоточное приложение, вычисляющее площадь угодий методом адаптивной квадратуры. При решении использовать парадигму рекурсивного параллелизма. Замечание: кривизну Земли из-за малой занимаемой площади не учитывать.

Составление​​ программы

Адаптивная квадратура, подсчет интеграла производится рекурсивным делением площади под кривой на две половины и потом ещё на две половины и так далее по рекурсии до совсем маленьких отрезков после чего суммируется. Для изменения f(x) необходимо изменить метод **double f(double x)**, который по сути просто возвращает необходимую f(x).

В программе выделены функции:

func и func1 считают правую и левую половину после разделения на два т.е. от (a, (а+b)/2), ( (a+b)/2, b).

Текст программы:

// Вариант 30

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <omp.h>  
  
using namespace std;  
  
double f(double x) {  
 return x + 2;  
}  
  
double coarse\_helper(double a, double b, double (f)(double)) {  
 return 0.5 \* (b - a) \* (f(a) + f(b));  
}  
  
double fine\_helper(double a, double b, double (f)(double)) {  
 double c = (a + b) / 2.0;  
 return 0.25 \* (b - a) \* (f(a) + 2 \* f(c) + f(b));  
}  
  
double\* a, \* b, \* c;  
int coun = 0, count1 = 0;  
double tolerance = 0.1;  
double fine, fine1;  
  
double func(double\* a, double\* b) {  
 double coarse = coarse\_helper(a[0], c[0], f);  
 fine = fine\_helper(a[0], c[0], f);  
 c[0] /= 2.0;  
 if ((fabs(coarse - fine) <= 3.0 \* tolerance) && (coun >= 1)) {  
 return fine;  
 }  
 else if (coun >= 10000) {  
 return NULL;  
 }  
 else {  
 ++coun;  
 return func(a, b);  
 }  
}  
  
double func1(double\* a, double\* b) {  
 double coarse = coarse\_helper(c[1], b[0], f);  
 fine1 = fine\_helper(c[1], b[0], f);  
 c[1] /= 2.0;  
  
 if ((fabs(coarse - fine1) <= 3.0 \* tolerance) && (count1 >= 1)) {  
 return NULL;  
 }  
 else if (count1 >= 10000) {  
 return NULL;  
 }  
 else {  
 ++count1;  
 return func1(a, b);  
 }  
}  
  
  
int main() {  
 a = new double[1];  
 b = new double[1];  
 c = new double[2];  
 cout << "Write a(left border):";  
 cin >> a[0];  
 cout << "Write b(right border):";  
 cin >> b[0];  
 c[0] = (a[0] + b[0]) / 2.0;  
 c[1] = (a[0] + b[0]) / 2.0;  
 int num[3];  
 for (int i = 0; i < 3; i++) num[i] = i;  
#pragma omp parallel sections  
 {  
#pragma omp section  
 {  
 func1(a, b);  
  
 }  
#pragma omp section  
 {  
 func(a, b);  
 }  
 }  
 cout << fine + fine1;  
 return 0;  
}