Асатрян Самвел Ваганович; БПИ 193-1; Вариант 29;

**Домашнее задание 3**

**Пояснительная записка**

**Текст задания:** Вычислить интеграл:

используя метод прямоугольников. Входные данные: вещественные числа a и b, функция f(x) задается с использованием описания в программе в виде отдельной функции. При суммировании использовать принцип дихотомии. Протестировать на различных функциях.

**Описание использованной модели вычислений:** Для вычислений использовался рекурсивный параллелизм. Для вычисления интеграла промежуток делится на две части и рекурсивно вызывается функция подсчета интеграла уже в этих двух частях.

**Список используемых источников для решения задачи:**

Модели параллелизма: [https://pro-prof.com/forums/topic/parallel-programming-paradigms#:~:text=%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B7%D0%BC%20%D0%BC%D0%BE%D0%B6%D0%B5%D1%82%20%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D1%8C%D1%81%D1%8F%2C%20%D0%BA%D0%BE%D0%B3%D0%B4%D0%B0,%D0%BD%D0%B0%D0%B4%20%D1%81%D0%B2%D0%BE%D0%B5%D0%B9%20%D1%87%D0%B0%D1%81%D1%82%D1%8C%D1%8E%20%D0%BE%D0%B1%D1%89%D0%B8%D1%85%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85](https://pro-prof.com/forums/topic/parallel-programming-paradigms#:~:text=%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D0%B8%D0%B7%D0%BC%20%D0%BC%D0%BE%D0%B6%D0%B5%D1%82%20%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D1%8C%D1%).

Метод дихотомии: <https://ru.wikibooks.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B4%D0%B8%D1%85%D0%BE%D1%82%D0%BE%D0%BC%D0%B8%D0%B8>

Материалы с SoftCraft: <http://www.softcraft.ru/edu/comparch/ref/par/>

**Описание файлов в папке проекта:**

1. HW\_3\_Integral.docx – исходник для пояснительной записки.
2. HW\_3\_Integral.pdf – пояснительная записка.
3. HW\_3\_Integral.cpp – файл с исходным кодом программы, пригодным для компиляции и запуска.
4. samples.png – скриншот работы программы с положительным значением

**Приложение**

Код программы

1. #include <cstdio> // include file for printf and perror
2. #include <cstdlib> // for the hostile example (rand function)
3. #include <cmath>
5. #include <errno.h>
6. #include <condition\_variable>
7. #include <vector>
8. #include <atomic>
10. #include <stack>
11. #include <thread>
12. #include <mutex>
13. #include <iostream>
14. # define PI     3.14159265358979323846
15. **using** **namespace** std;

18. **typedef** **double** real;


22. **struct** Task{
23. real (\*f)(real);
24. real a, b, dx;
25. real maxRecDepth;
26. Task(){}
27. Task(real (\*f)(real), real a, real b, real dx,**int** rec):f(f),a(a),b(b),
28. dx(dx),maxRecDepth(rec){}
30. real execute(Task & task1, Task & task2, real & integral){
32. real m   = (a + b)/2,  h   = (b - a)/2;
34. **if** (maxRecDepth <= 0 || 2\*h < dx){
35. integral = f(m)\*(b-a);
36. **return** **true**;
37. }
39. task1 = Task(f, a, m, dx, maxRecDepth-1);
40. task2 = Task(f, m, b, dx, maxRecDepth-1);
41. **return** **false**;
42. }
43. };
45. **class** IntegrationEngine{
46. **public**:
47. IntegrationEngine(**int** n\_threads=-1){
48. **if** (n\_threads<0)
49. n\_threads = **thread**::hardware\_concurrency();
50. running\_tasks = 0;
51. pending\_tasks = 0;
52. **for** (**int** i = 0; i < n\_threads; i++){
53. threads.push\_back(std::**thread**(&IntegrationEngine::infinite\_loop,**this**));
54. }
56. };
58. real integrate(real (\*f)(real),     // function ptr to integrate
59. real a, real b,      // interval [a,b]
60. real dx,         // step size
61. **int** maxRecDepth) {     // recursion cap
62. **if** (b == a) **return** 0;
64. integral = 0;
66. std::unique\_lock<std::mutex> lock(mx\_for\_stack);
67. tasks.emplace(f, a, b, dx,maxRecDepth);
68. pending\_tasks++;
69. lock.unlock();

72. wait();
73. **return** integral;
75. }
76. **void** infinite\_loop(){
77. **while**(**true**){
78. std::unique\_lock<std::mutex> lock(mx\_for\_stack);
79. **if** (pending\_tasks==0){
80. locker.wait(lock, [**this**] {
81. **return** pending\_tasks>0 || stop\_threads; });
82. }
83. **if** (stop\_threads){
84. **break**;
85. }
87. running\_tasks++;
88. Task task = (tasks.top());
89. tasks.pop();
90. pending\_tasks--;
91. lock.unlock();


95. real result=0;
96. Task task1, task2;
97. **bool** res = task.execute(task1, task2, result);
98. **if** (res){
99. std::unique\_lock<std::mutex> lock2(mx\_for\_integral);
100. integral+= result;
101. lock2.unlock();
102. }
103. **else**{
104. std::unique\_lock<std::mutex> lock3(mx\_for\_stack);
105. tasks.push(task1);
106. tasks.push(task2);
107. pending\_tasks+=2;
108. lock3.unlock();
109. }
111. locker.notify\_all();
113. running\_tasks--;
114. std::unique\_lock<std::mutex> lock4(mx\_for\_waiter);
115. waiter.notify\_all();
116. lock4.unlock();
118. }
119. }
120. **void** wait(){
121. std::unique\_lock<std::mutex> lock(mx\_for\_waiter);
122. locker.notify\_one();
123. waiter.wait(lock, [**this**] {
124. **return** running\_tasks==0 && pending\_tasks==0;});
126. }
127. **void** join(){
128. stop\_threads = **true**;
129. std::unique\_lock<std::mutex> lock(mx\_for\_stack);
130. locker.notify\_all();
131. lock.unlock();
132. **for** (auto& t : threads){
133. t.join();
134. }
135. }
136. ~IntegrationEngine(){
137. join();
138. }

141. **private**:
142. std::vector<std::**thread**> threads;
143. stack<Task> tasks;
144. atomic<**int**> running\_tasks;
145. atomic<**int**> pending\_tasks;
147. std::mutex mx\_for\_stack;
148. std::mutex mx\_for\_integral;
149. std::mutex mx\_for\_waiter;
150. std::condition\_variable locker;
151. std::condition\_variable waiter;
153. real integral;
154. **bool** stop\_threads = **false**;
155. };


159. real std\_normal\_pdf(real x){
160. **return** exp(-x\*x/2)/sqrt(2 \* PI);
161. }
163. real sample\_1(real x) {
164. **return** 5\*x\*x + 10\*x +7;
165. }
167. real sample\_2(real x){
168. **return** sin(x) + cos(x);
169. }

172. **int** main() {
173. **int** threadNumbers = **thread**::hardware\_concurrency();
174. real I=0;
175. IntegrationEngine engine(threadNumbers);

178. // Let I be the integral of sin(x) from 0 to 20
179. I = engine.integrate(sin, 0, 20, 1e-3, 15);
180. printf("integrate(sin, 0, 20) = %0.8lf\n", I);   // print the result
182. // Gaussian integrals
183. I = engine.integrate(std\_normal\_pdf, -1, 1, 1e-3, 15);
184. printf("\nintegrate(Normal pdf, -1,1) = %0.8lf\n", I);
186. I = engine.integrate(sample\_1, -5, 10, 1e-3, 15);
187. printf("\nintegrate(5x^2+10x+7,-5,10) = %0.8lf\n", I);
189. I = engine.integrate(sample\_2, -1, 1, 1e-3, 15);
190. printf("\nintegrate(sin(x)+cos(x), -1, 1) = %0.8lf\n", I);

193. **return** 0;
194. }