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

Домашнее задание 3

Пояснительная записка

Текст задания: Вычислить интеграл:

$$\int_a^b f(x)\,dx$$

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.

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

```
#include <cstdio> // include file for printf and perror
#include <cstdlib> // for the hostile example (rand function)
#include <cmath>

#include <errno.h>
#include <condition_variable>
#include <vector>
#include <atomic>

#include <stack>
#include <thread>
#include <mutex>
#include <iostream>
# define PI     3.14159265358979323846
using namespace std;


typedef double real;



struct Task{
    real (*f)(real);
    real a, b, dx;
    real maxRecDepth;
    Task(){}
    Task(real (*f)(real), real a, real b, real dx,int rec):f(f),a(a),b(b),
        dx(dx),maxRecDepth(rec){}

    real execute(Task & task1, Task & task2, real & integral){

        real m   = (a + b)/2,  h   = (b - a)/2;

        if (maxRecDepth <= 0 || 2*h < dx){
            integral = f(m)*(b-a);
```

```
            return true;
        }

        task1 = Task(f, a, m, dx, maxRecDepth-1);
        task2 = Task(f, m, b, dx, maxRecDepth-1);
        return false;
    }
};

class IntegrationEngine{
public:
    IntegrationEngine(int n_threads=-1){
        if (n_threads<0)
            n_threads = thread::hardware_concurrency();
        running_tasks = 0;
        pending_tasks = 0;
        for (int i = 0; i < n_threads; i++){
            threads.push_back(std::thread(&IntegrationEngine::infinite_loop,this));
        }

    };

    real integrate(real (*f)(real),     // function ptr to integrate
                        real a, real b,      // interval [a,b]
                        real dx,         // step size
                        int maxRecDepth) {     // recursion cap
        if (b == a) return 0;

        integral = 0;

        std::unique_lock<std::mutex> lock(mx_for_stack);
        tasks.emplace(f, a, b, dx,maxRecDepth);
        pending_tasks++;
        lock.unlock();


        wait();
        return integral;

    }
    void infinite_loop(){
        while(true){
            std::unique_lock<std::mutex> lock(mx_for_stack);
            if (pending_tasks==0){
                locker.wait(lock, [this] {
                    return pending_tasks>0 || stop_threads; });
            }
            if (stop_threads){
                break;
            }

            running_tasks++;
            Task task = (tasks.top());
            tasks.pop();
            pending_tasks--;
            lock.unlock();



            real result=0;
            Task task1, task2;
            bool res = task.execute(task1, task2, result);
            if (res){
                std::unique_lock<std::mutex> lock2(mx_for_integral);
                       integral+= result;
                       lock2.unlock();
                  }
                  else{
                      std::unique_lock<std::mutex> lock3(mx_for_stack);
```

```
                    tasks.push(task1);
                    tasks.push(task2);
                    pending_tasks+=2;
                    lock3.unlock();
                }

                locker.notify_all();

                running_tasks--;
                std::unique_lock<std::mutex> lock4(mx_for_waiter);
                waiter.notify_all();
                lock4.unlock();

            }
        }
        void wait(){
            std::unique_lock<std::mutex> lock(mx_for_waiter);
            locker.notify_one();
            waiter.wait(lock, [this] {
                return running_tasks==0 && pending_tasks==0;});

        }
        void join(){
            stop_threads = true;
            std::unique_lock<std::mutex> lock(mx_for_stack);
            locker.notify_all();
            lock.unlock();
            for (auto& t : threads){
                t.join();
            }
        }
        ~IntegrationEngine(){
            join();
        }


    private:
        std::vector<std::thread> threads;
        stack<Task> tasks;
        atomic<int> running_tasks;
        atomic<int> pending_tasks;

        std::mutex mx_for_stack;
        std::mutex mx_for_integral;
        std::mutex mx_for_waiter;
        std::condition_variable locker;
        std::condition_variable waiter;

        real integral;
        bool stop_threads = false;
    };



    real std_normal_pdf(real x){
        return exp(-x*x/2)/sqrt(2 * PI);
    }

    real sample_1(real x) {
        return 5*x*x + 10*x +7;
    }

    real sample_2(real x){
        return sin(x) + cos(x);
    }


    int main() {
        int threadNumbers = thread::hardware_concurrency();
```

```
174.            real I=0;
175.            IntegrationEngine engine(threadNumbers);
176.
177.
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
181.
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);
185.
186.            I = engine.integrate(sample_1, -5, 10, 1e-3, 15);
187.            printf("\nintegrate(5x^2+10x+7,-5,10) = %0.8lf\n", I);
188.
189.            I = engine.integrate(sample_2, -1, 1, 1e-3, 15);
190.            printf("\nintegrate(sin(x)+cos(x), -1, 1) = %0.8lf\n", I);
191.
192.
193.            return 0;
194.        }
```