ИДЗ-4

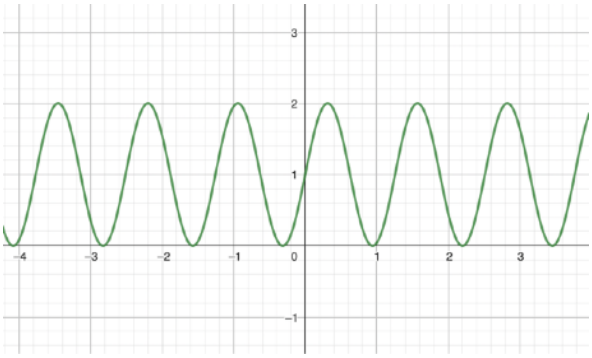
Баханкова Дарья Сергеевна
БПИ215
Вариант 34

https://github.com/Dashbah/Bakhankova_Asm_HW4

34. ***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.*

Предисловие

Будем считать, что функция нам дана (f(x)=sin(5 x) +1) и площадь ограничена снизу осью OX. Входные параметры - левая и правая границы. Некорректными считаются случаи, когда левая граница больше правой.



Задача решена с помощью вычисления интеграла методом адаптивной квадратуры.

Исходя из требований к заданию, оно выполнено на **8 баллов**

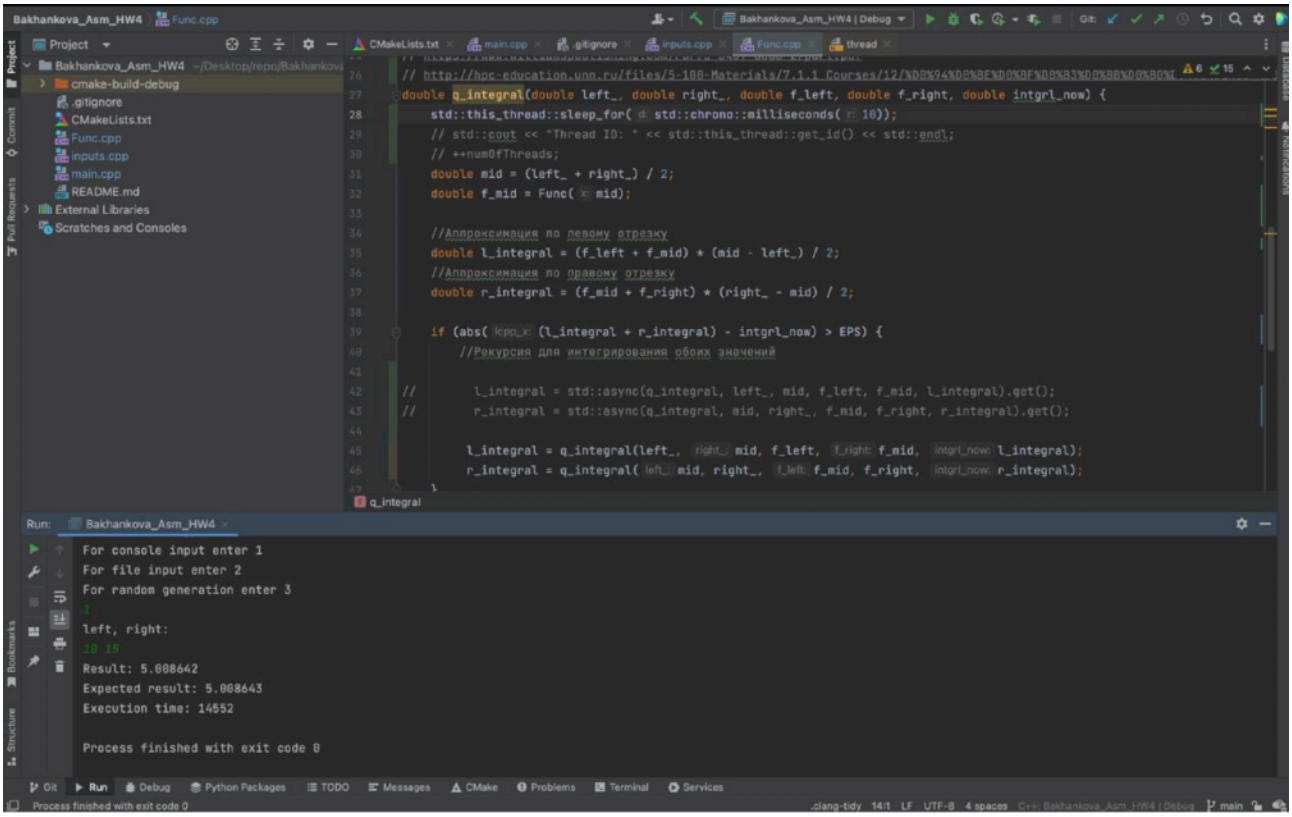
Метод адаптивной квадратуры:

Интервал разбивается на две части и для каждой из них рекурсивно вычисляется интеграл. Их вызовы независимы, т.е. каждый из них работает над своей частью общих данных. Значит, их можно выполнить параллельно.

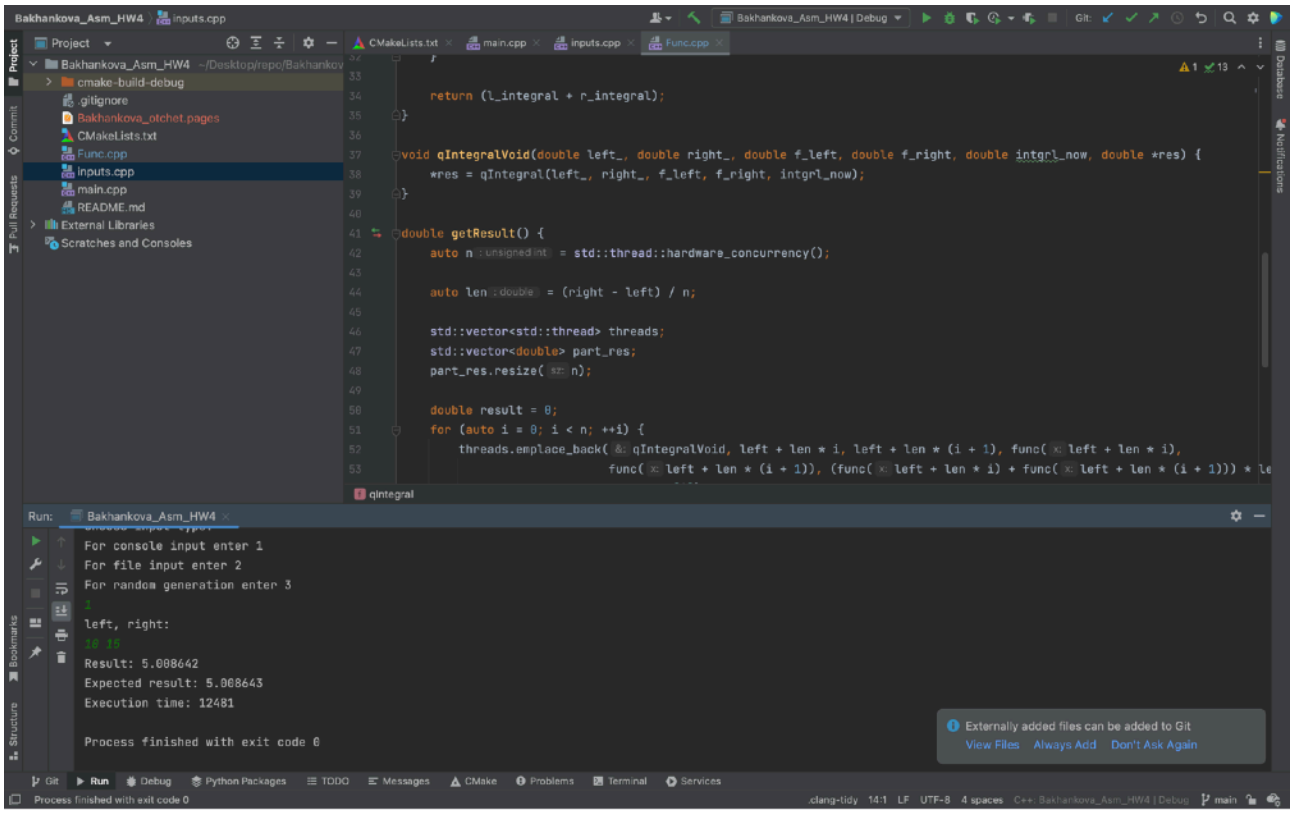
В некоторых источниках предлагается на каждом вызове рекурсии параллелить вызовы для левой и правой части интервала. То есть эти (на скриншоте) функции вывести в два потока. Но это решение сомнительное, так как возможна некорректная работа рекурсии плюс у нас ограничено количество потоков.

```cpp
double qIntegral(double left_, double right_, double f_left, double f_right, double intgrl_now) {
    std::this_thread::sleep_for( d: std::chrono::milliseconds( r: 10));
    double mid = (left_ + right_) / 2;
    double f_mid = func( x: mid);

    //Аппроксимация по левому отрезку
    double l_integral = (f_left + f_mid) * (mid - left_) / 2;
    //Аппроксимация по правому отрезку
    double r_integral = (f_mid + f_right) * (right_ - mid) / 2;

    if (abs( lcpp_x: (l_integral + r_integral) - intgrl_now) > EPS) {
        //Рекурсия для интегрирования обоих значений
        l_integral = qIntegral(left_,  right_: mid, f_left,  f_right: f_mid,  intgrl_now: l_integral);
        r_integral = qIntegral( left_: mid, right_,  f_left: f_mid, f_right,  intgrl_now: r_integral);
    }

    return (l_integral + r_integral);
}
```

Поскольку у нас (у меня) 8 ядер, разделим интервал на 8 частей и для каждой посчитаем интеграл параллельно. Это не только ускоряет время выполнения, но и уменьшает глубину рекурсии. Как известно, рекурсия при большой глубине может привести, например, к переполнению стека.
Но выполнение программы будет не в 8 раз быстрее, на малых данных даже дольше, потому что вызов потока это долгий процесс. Чтобы увидеть разницу в производительности, поставим засыпание на 10 мс на каждом вызове рекурсии в обеих программах (однопоточная и многопоточная). Разница во времени на скриншотах:



Один поток (14552мс)



8 потоков(12481мс)

Источники:

http://templet.ssau.ru/wiki/_media/presentations/%D0%B2%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F.pdf

https://studfile.net/preview/16404441/page:6/