

Домашнее задание №3  
Практические приемы построения  
многопоточных приложений  
Вариант 30

Арыслан Якшибаев, БПИ196

November 17, 2020

## Contents

1	Задание	1
2	Модель вычислений: рекурсивный параллелизм	1
3	Сборка и выполнение	1
3.1	Сборка . . . . .	1
3.2	Запуск . . . . .	1
3.2.1	Примеры запуска . . . . .	1
3.2.2	Скриншоты . . . . .	2
4	Исходный код	3

## 1 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 ограничена 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.

## 2 Модель вычислений: рекурсивный параллелизм

Рекурсивный параллелизм подразумевает, что рекурсивные вызовы выполняются асинхронно. Это выполнимо только когда параллельные потоки независимы (т.е. они либо не разделяют никаких общих переменных, либо используют их только на чтение). В данной работе рекурсивный параллелизм реализован с использованием функций библиотеки `pthread`, объявленных в header-файле `<future>`. Данный подход подробно описан [здесь \(ссылка\)](#).

## 3 Сборка и выполнение

### 3.1 Сборка

```
g++ -pthread main.cpp -o main
```

### 3.2 Запуск

Программа принимает на вход данные через аргументы командной строки, и выводит вычисленный результат в `stdout`.

```
./main <left_bound> <right_bound> [-f <function_number>]
```

- `left_bound` - левая граница (например, 1.)

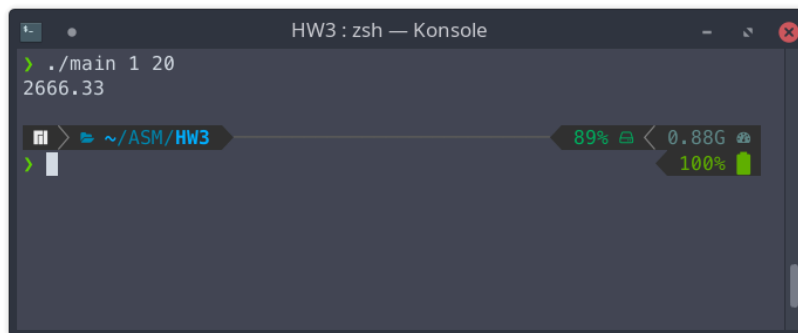
- `right_bound` - правая граница (например, 100 .)
- `function_number` (опционально) - номер функции  $f(x)$  . Допустимые значения: 1, 2, 3, 4, 5 . **Значение по умолчанию: 1.**

### 3.2.1 Примеры запуска

```
./main 1 100
```

```
./main 10 1000 -f 2
```

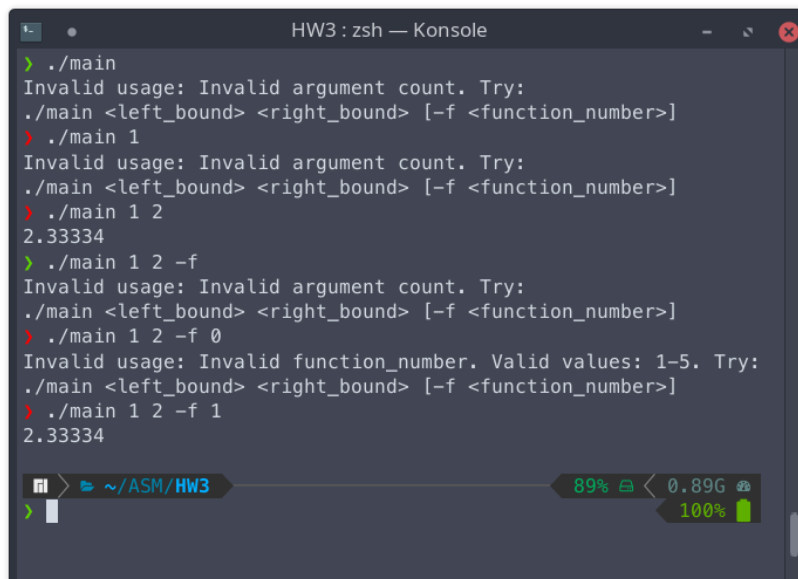
### 3.2.2 Скриншоты



```

HW3: zsh — Konsole
> ./main 1 20
2666.33
>

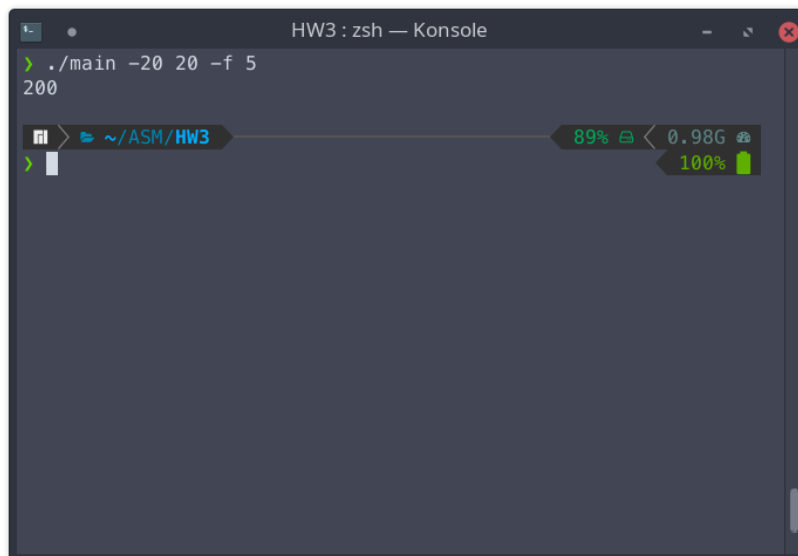
```



```

HW3: zsh — Konsole
> ./main
Invalid usage: Invalid argument count. Try:
./main <left_bound> <right_bound> [-f <function_number>]
> ./main 1
Invalid usage: Invalid argument count. Try:
./main <left_bound> <right_bound> [-f <function_number>]
> ./main 1 2
2.33334
> ./main 1 2 -f
Invalid usage: Invalid argument count. Try:
./main <left_bound> <right_bound> [-f <function_number>]
> ./main 1 2 -f 0
Invalid usage: Invalid function_number. Valid values: 1-5. Try:
./main <left_bound> <right_bound> [-f <function_number>]
> ./main 1 2 -f 1
2.33334
>

```



## 4 Исходный код

```
#include <cmath>
#include <ctime>
#include <future>
#include <iostream>
#include <sstream>
#include <stdexcept>

using namespace std;

// Точность вычислений
#define EPSILON 1e-6

// Объявление типа для удобного создания массива функций
typedef double (*DoubleFunctionWithOneParameter)(double a);

// Математические функции
double f1(double x) { return x * x; }
double f2(double x) { return x; }
double f3(double x) { return sin(x); }
double f4(double x) { return tan(x); }
double f5(double x) { return 2 * x + 5; }

// Основная рекурсия.
// Аргументы:
// double left, right: границы интервала по оси X.
// double fleft, fright: значения функции f(x) на границах интервала (передаём
// эти значения в рекурсию чтобы не считать их лишней раз)
// double larea: приблизительное значение площади,
```

```

// вычисленное методом трапеций на предыдущем шагу рекурсии.
// double f(double x): функция f(x), относительно которой считаются значения.
double quad(double left, double right, double fleft, double fright,
             double lrarea, double f(double)) {
    double mid = (left + right) / 2; // Находим среднюю точку интервала
    double fmid = f(mid); // Находим значение функции в средней точке
    double larea = (fleft + fmid) * (mid - left) /
        2; // Находим приблизительную площадь левой части интервала
        // (метод трапеций)
    double rarea = (fmid + fright) * (right - mid) /
        2; // Находим приблизительную площадь правой части
    if (abs(larea + rarea - lrarea) >
        EPSILON) { // Если разница между значением, вычисленным на предыдущем шагу
        // и на текущем больше, чем заданная точность, уточняем.
        future<double> larea_promise =
            async(&quad, left, mid, fleft, fmid, larea,
                f); // Рекурсивно считаем площадь левой половины
        future<double> rarea_promise =
            async(&quad, mid, right, fmid, fright, rarea,
                f); // Рекурсивно считаем площадь правой половины
        // Ждём выполнения асинхронной рекурсии и получаем результаты.
        larea = larea_promise.get();
        rarea = rarea_promise.get();
    }
    return rarea + larea; // Возвращаем полученное значение.
}

// Функция-хелпер для более очевидного вызова рекурсии.
double quad(double left, double right, double f(double)) {
    return quad(left, right, f(left), f(right),
        (f(left) + f(right)) * (right - left) / 2, f);
}

int main(int argc, char *argv[]) {
    double a, b;
    int fn = 0;
    DoubleFunctionWithOneParameter funcs[] = {f1, f2, f3, f4, f5};

    // Парсим аргументы командной строки
    try {
        if (argc != 3 && (argc != 5))
            throw invalid_argument("Invalid argument count");
        stringstream ss;
        for (int i = 1; i < argc; ++i)
            ss << argv[i] << " ";
        ss.flush();
        ss.seekg(0);
        ss >> a >> b;
        if (argc == 5) {
            string tmp;

```

```

    ss >> tmp;
    ss >> fn;
    fn--;
    if (fn < 0 || fn > 4)
        throw invalid_argument("Invalid function_number. Valid values: 1-5");
    }
} catch (exception &e) {
    cerr << "Invalid usage: " << e.what() << ". Try:" << endl;
    cerr << argv[0] << " <left_bound> <right_bound> [-f <function_number>]"
        << endl;
    return -1;
}

// Вызов рекурсии и вывод ответа
cout << quad(a, b, funcs[fn]) << endl;
return 0;
}

```