

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук
Образовательная программа бакалавриата «Программная инженерия»

**ОТЧЁТ К ДОМАШНЕМУ ЗАДАНИЮ № 3
ПО ДИСЦИПЛИНЕ
«АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ»**

ВАРИАНТ 15

Исполнитель
студент группы БПИ191
Н.К. Игумнов

Москва 2020

ЗАДАНИЕ

Вывести список всех целых чисел, содержащих от 4 до 9 значащих цифр, которые после умножения будут содержать все те же самые цифры в произвольной последовательности и в произвольном количестве. Входные данные: целое положительное число n , больше единицы и меньше десяти. Количество потоков является входным параметром.

ИНТЕРПРЕТАЦИЯ

Пройдёмся по всем числам x из промежутка $[1000, 10^{10})$, если после умножения на входной параметр $n \in (1, 10)$ все цифры числа $new_value = x \cdot n$ содержатся в x , то добавим его в итоговый ответ. Выведем список всех полученных чисел в консоль / файл.

РЕШЕНИЕ

Для проверки, содержатся ли все цифры нового числа *new_value* в *x*, было предложено пять вариантов:

1. Приведём изначальное целочисленное число *x* в строковое методом *std::to_string()*, пройдемся по всем символам *y* в полученной строке и добавим их в *std::set<char> s* (сделаем *s.insert(y)*). Повторим эту операцию для $x \cdot n$ (только выделим для данного числа другой *set*). Если *set*-ы совпадут, то условие выполняется (работает за ≈ 307.704 секунд для $n = 2$, *thread_number* = 4, разрядов до 8, Debug).
2. Пройдемся по всем цифрам *y* числа *x* и добавим их в *std::set<int> s* (сделаем *s.insert(y)*). Повторим эту операцию для $x \cdot n$ (только выделим для данного числа другой *set*). Если *set*-ы совпадут, то условие выполняется (работает за ≈ 262.573 секунд для $n = 2$, *thread_number* = 4, разрядов до 8, Debug).
3. Выделим для каждого числа (*x* и $x \cdot n$) отдельный массив *bool*-ов размера 10 (изначально заполненный *false*), отвечающий за наличие / отсутствие конкретных цифр в числе. Пройдемся по всем цифрам *y* числа *x* и обновим значение массива *a*, соответствующее данной цифре (сделаем *a[y] = true*). Повторим эту операцию для $x \cdot n$. Если массивы совпадут, то условие выполняется (работает за ≈ 15.06 секунд для $n = 2$, *thread_number* = 4, разрядов до 8, Debug).
4. Выделим для каждого числа отдельный *std::bitset<10>*, отвечающий за наличие / отсутствие конкретных цифр в числе. Пройдемся по всем цифрам *y* числа *x* и обновим значение *bitset*-а *b*, соответствующее данной цифре (сделаем *b[y] = true*). Повторим эту операцию для $x \cdot n$. Если *bitset*-ы совпадут, то условие выполняется (работает за ≈ 20.87 секунд для $n = 2$, *thread_number* = 4, разрядов до 8, Debug).
5. Выделим для каждого числа отдельное дополнительное целое число *f_y (int)*, отвечающее за наличие / отсутствие конкретных цифр в числе. Пройдемся по всем цифрам *y* числа *x* и обновим *f_y*. Повторим эту операцию для $x \cdot n$. Если данные числа совпадут, то условие выполняется (работает за ≈ 12.436 секунд для $n = 2$, *thread_number* = 4, разрядов до 8, Debug). – Почему это возможно? В худшем случае число состоит из всех цифр 0, ..., 9. Тогда значение этой функции будет равно $2^0 + \dots + 2^9 = 2^{10} - 1 = 1023$. То есть это число помещается в *int*.

Для того, чтобы программа сгенерировала входные числа, требуется раскомментировать 11-ую строчку: `#define GENERATE`.

Для того, чтобы программа выводила всю информацию в файл, требуется раскомментировать 12-ую строчку: `#define FILE_OUT`.

Функция *Read* считывает входные данные: n – входной множитель, *thread_number* – количество потоков и, если требуется, путь до выходного файла *file_name*.

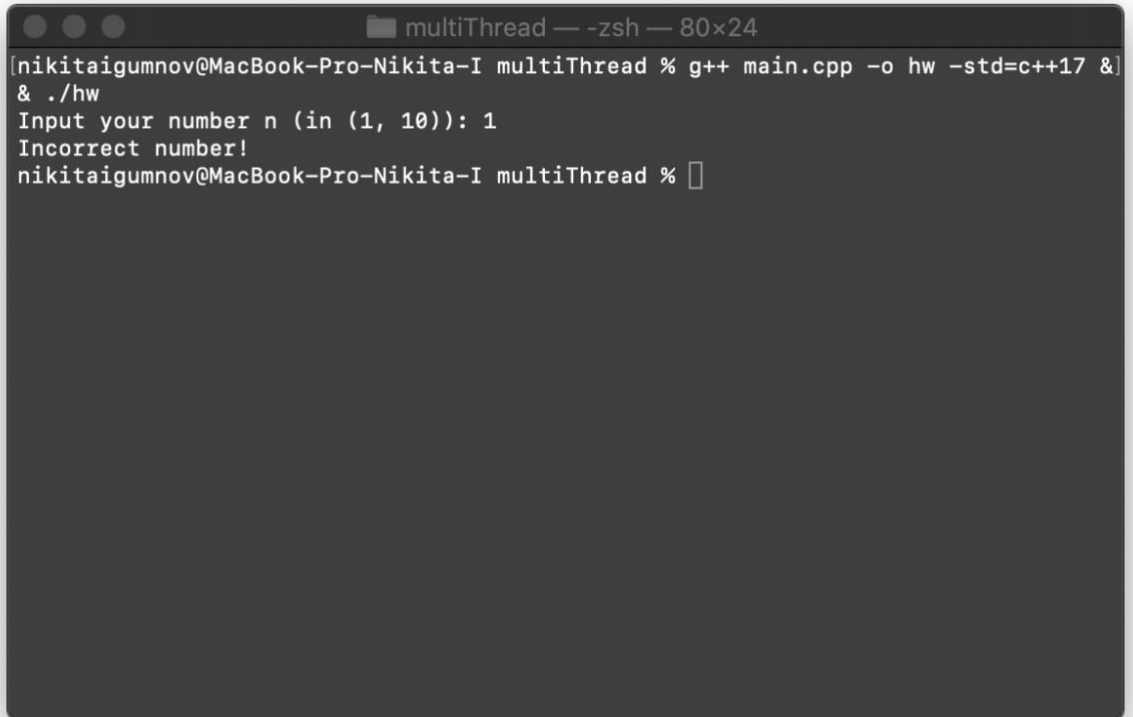
В данной задаче я применяю модель **итеративного параллелизма**.

Функция *Compute* находит ответ на задачу, вызывая *thread_number* потоков. Каждый из потоков описывается итеративной функцией *ComputeThread* для своего промежутка. В этой функции для каждого числа из данного промежутка будет проверяться условие задачи с помощью функции *Fun* – возвращающей структуру данных, отвечающую за наличие / отсутствие конкретных цифр в числе (см. выше).

Функция *Print* выводит в консоль / выходной файл (в зависимости от `FILE_OUT`) список всех полученных чисел.

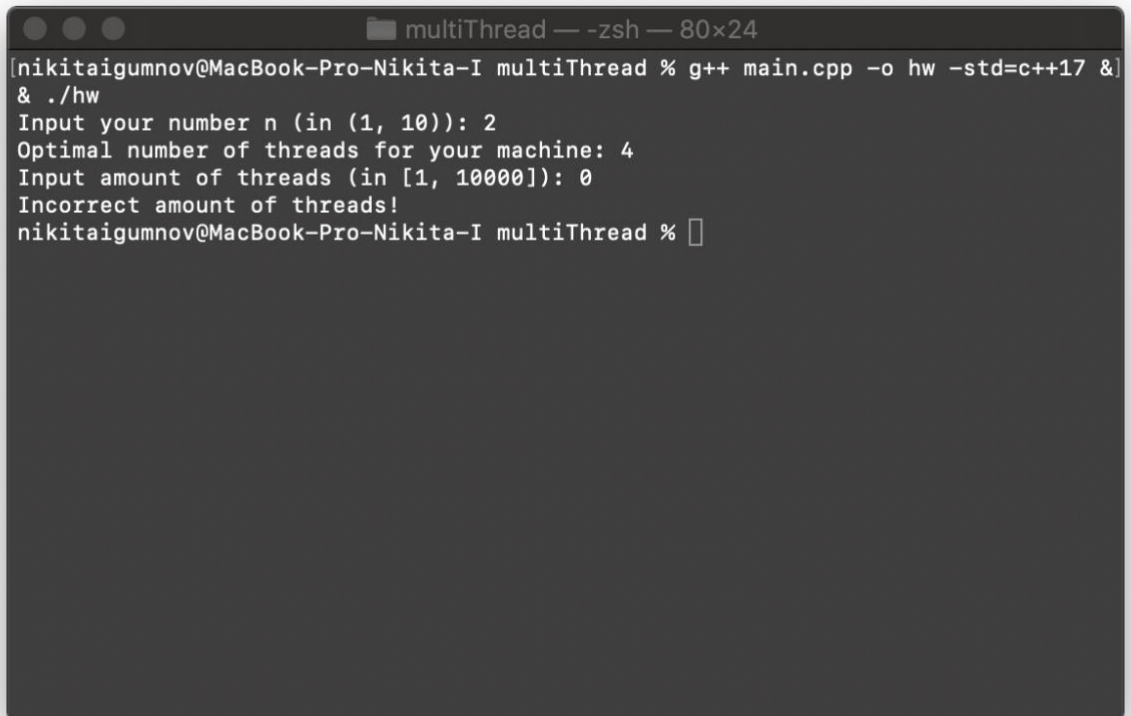
ТЕСТИРОВАНИЕ ПРОГРАММЫ

1. Некорректный ввод ($N = 1$)



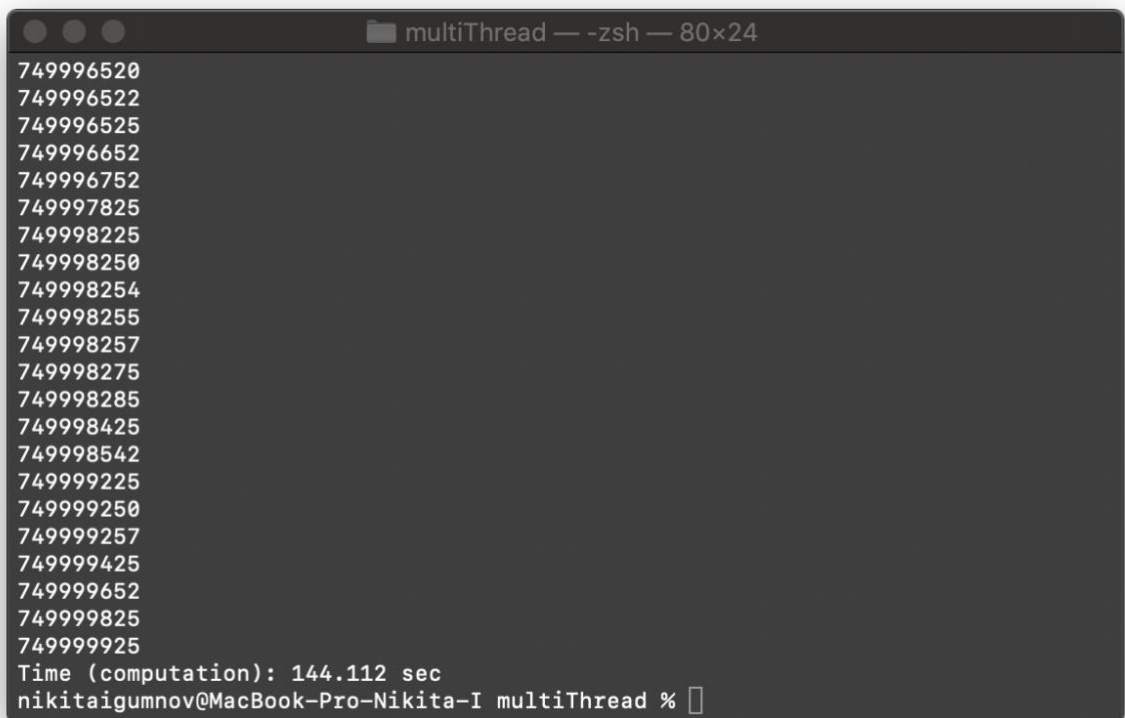
```
multiThread — -zsh — 80x24
[nikitaigumnov@MacBook-Pro-Nikita-I multiThread % g++ main.cpp -o hw -std=c++17 &]
& ./hw
Input your number n (in (1, 10)): 1
Incorrect number!
nikitaigumnov@MacBook-Pro-Nikita-I multiThread %
```

Программа отработала успешно.

2. Некорректный ввод ($N = 2, thread_number = 0$)

```
multiThread — -zsh — 80x24
[nikitaigumnov@MacBook-Pro-Nikita-I multiThread % g++ main.cpp -o hw -std=c++17 &]
& ./hw
Input your number n (in (1, 10)): 2
Optimal number of threads for your machine: 4
Input amount of threads (in [1, 10000]): 0
Incorrect amount of threads!
nikitaigumnov@MacBook-Pro-Nikita-I multiThread %
```

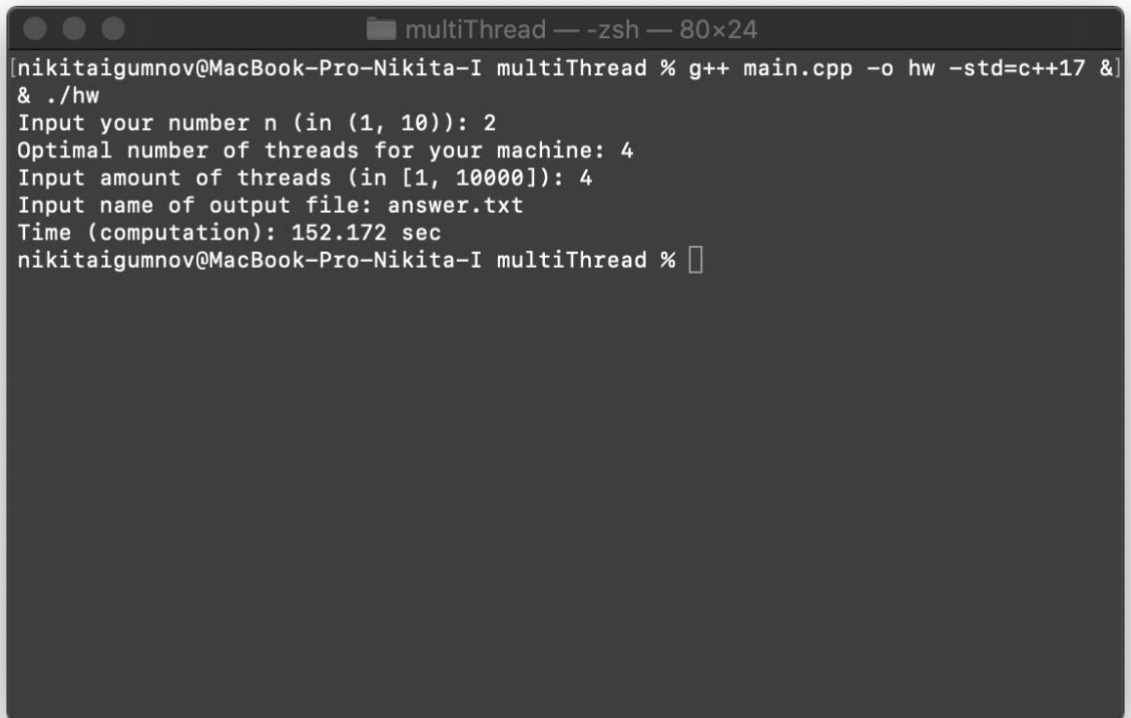
Программа отработала успешно.

3. $N = 7, thread_number = 4$ 

```
multiThread — -zsh — 80x24
749996520
749996522
749996525
749996652
749996752
749997825
749998225
749998250
749998254
749998255
749998257
749998275
749998285
749998425
749998542
749999225
749999250
749999257
749999425
749999652
749999825
749999925
Time (computation): 144.112 sec
nikitaigumnov@MacBook-Pro-Nikita-I multiThread %
```

Программа отработала успешно.

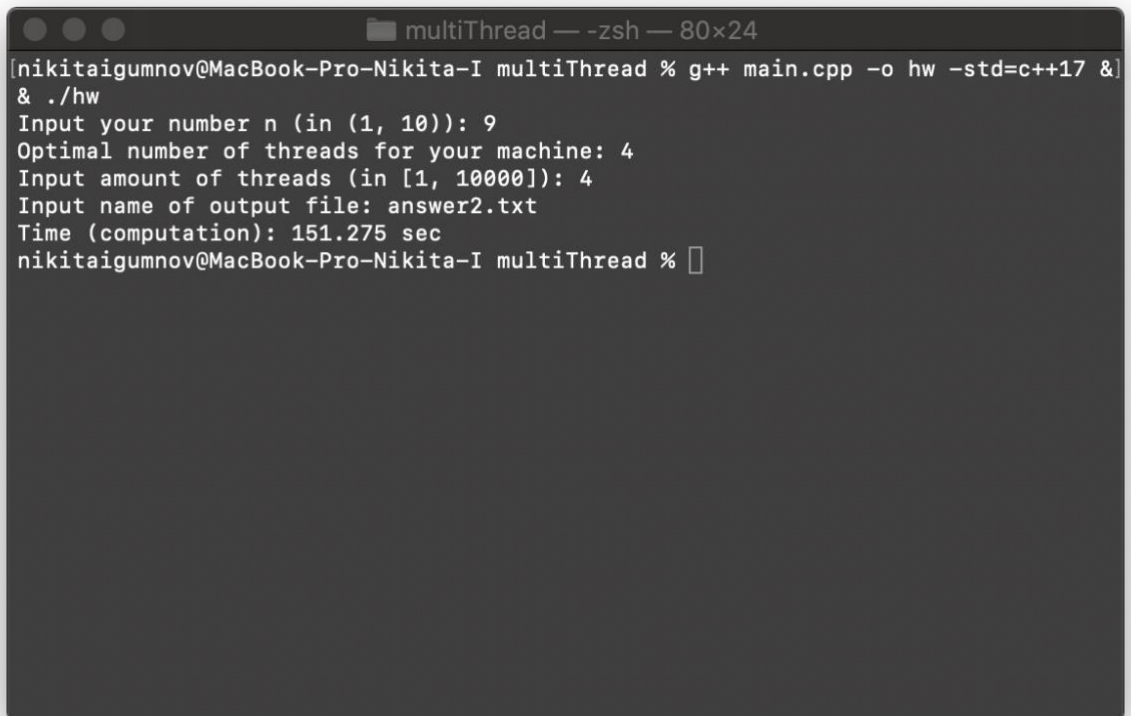
4. $N = 2, thread_number = 4$ (см. tests/answer.txt)



```
multiThread — -zsh — 80x24
[nikitaigumnov@MacBook-Pro-Nikita-I multiThread % g++ main.cpp -o hw -std=c++17 &]
& ./hw
Input your number n (in (1, 10)): 2
Optimal number of threads for your machine: 4
Input amount of threads (in [1, 10000]): 4
Input name of output file: answer.txt
Time (computation): 152.172 sec
nikitaigumnov@MacBook-Pro-Nikita-I multiThread %
```

Программа отработала успешно.

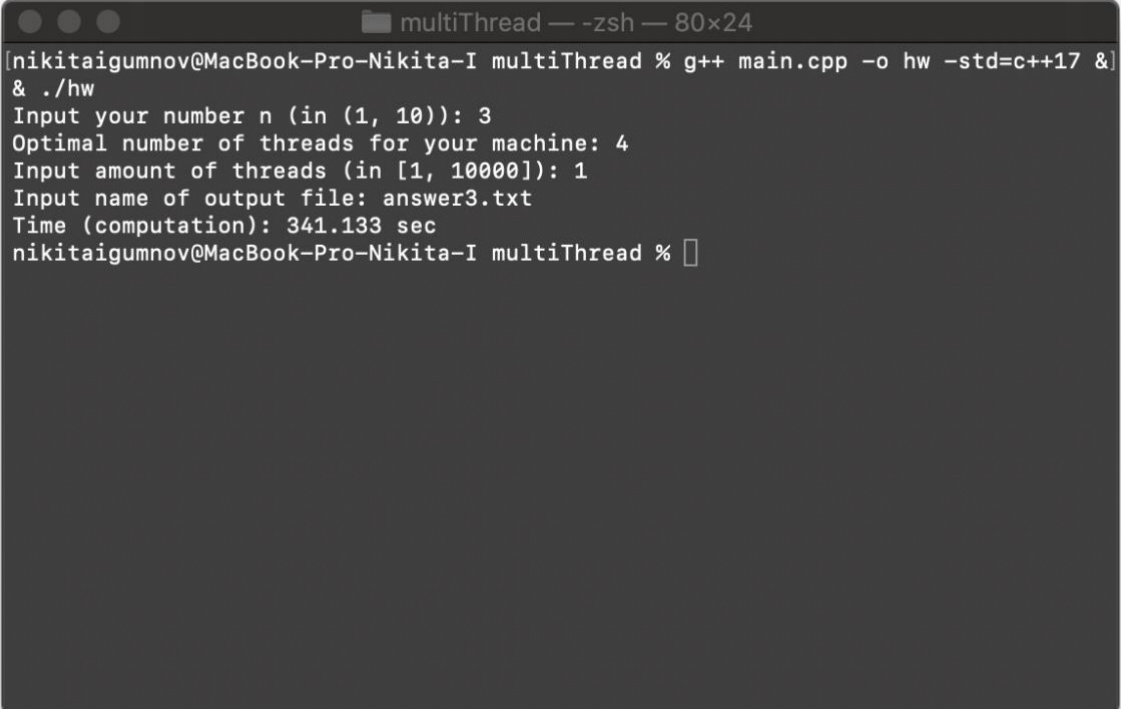
5. $N = 9, thread_number = 4$ (см. tests/answer2.txt)



```
multiThread — -zsh — 80x24
[nikitaigumnov@MacBook-Pro-Nikita-I multiThread % g++ main.cpp -o hw -std=c++17 &]
& ./hw
Input your number n (in (1, 10)): 9
Optimal number of threads for your machine: 4
Input amount of threads (in [1, 10000]): 4
Input name of output file: answer2.txt
Time (computation): 151.275 sec
nikitaigumnov@MacBook-Pro-Nikita-I multiThread %
```

Программа отработала успешно

6. $N = 3$, $thread_number = 1$ (см. tests/answer3.txt)



```
multiThread — -zsh — 80x24
[nikitaigumnov@MacBook-Pro-Nikita-I multiThread % g++ main.cpp -o hw -std=c++17 &]
& ./hw
Input your number n (in (1, 10)): 3
Optimal number of threads for your machine: 4
Input amount of threads (in [1, 10000]): 1
Input name of output file: answer3.txt
Time (computation): 341.133 sec
nikitaigumnov@MacBook-Pro-Nikita-I multiThread %
```

Программа отработала успешно

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Ошибки многопоточности в C++ [Электронный ресурс]. Режим доступа: <https://otus.ru/nest/post/145/>, свободный. (дата обращения: 17.11.2020)
- 2) Simple example of threading in C++ [Электронный ресурс]. Режим доступа: <https://stackoverflow.com/questions/266168/simple-example-of-threading-in-c>, свободный. (дата обращения: 17.11.2020)
- 3) std::thread::hardware_concurrency [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/thread/thread/hardware_concurrency, свободный. (дата обращения: 17.11.2020)
- 4) Measuring execution time of a fuction in C++ [Электронный ресурс]. Режим доступа: <https://stackoverflow.com/questions/22387586/measuring-execution-time-of-a-function-in-c>, свободный. (дата обращения: 17.11.2020)
- 5) Практические приёмы построения многопоточных приложений [Электронный ресурс]. Режим доступа: <http://www.softcraft.ru/edu/comparch/tasks/t03/>, свободный. (дата обращения: 17.11.2020)
- 6) Choosing the number of threads at runtime [Электронный ресурс]. Режим доступа: <https://livebook.manning.com/book/c-plus-plus-concurrency-in-action/chapter-2/92/>, свободный. (дата обращения: 17.11.2020)

КОД ПРОГРАММЫ

Смотрите multiThread.cpp

```
#include <iostream>
#include <optional>
#include <thread>
#include <chrono>
#include <random>
#include <fstream>
#include <list>
#include <vector>
#include <mutex>

// #define GENERATE
// #define FILE_OUT

constexpr int64_t MIN_VALUE = 1000;
constexpr int64_t MAX_VALUE = 999'999'99911;

std::mutex mut;

bool Read(int *n, int *thread_number, std::string *file_name) { // NOLINT
#ifdef GENERATE
    std::mt19937
gen(std::chrono::high_resolution_clock::now().time_since_epoch().count());
    std::uniform_int_distribution<int> dist_n(2, 9);
    *n = dist_n(gen);
    std::uniform_int_distribution<int64_t> dist_t(1, MAX_VALUE - MIN_VALUE +
1);
    *thread_number = dist_t(gen);
    *file_name = "answer.txt";
    std::cout << "Number n: " << *n << '\n';
    std::cout << "Amount of threads: " << *thread_number << '\n';
#else
    std::cout << "Input your number n (in (1, 10)):";
    std::cin >> *n;
    if (*n <= 1 || *n >= 10) {
        std::cout << "Incorrect number!\n";
        return false;
    }
    // see https://livebook.manning.com/book/c-plus-plus-concurrency-in-action/chapter-2/92 for details
    unsigned int hardware_threads = std::thread::hardware_concurrency();
    std::cout << "Optimal number of threads for your machine: " <<
(hardware_threads != 0 ? hardware_threads : 2)
    << '\n';
    std::cout << "Input amount of threads (in [1, 10000]): ";
    std::cin >> *thread_number;
    if (*thread_number < 1 || *thread_number > 10000) {
        std::cout << "Incorrect amount of threads!\n";
        return false;
    }
#endif
#ifdef FILE_OUT
    std::cout << "Input name of output file: ";
    std::cin >> *file_name;
    std::ofstream out(*file_name);
    if (!out.is_open()) {
        std::cout << "Incorrect file name!\n";
    }
#endif
}
```

```

        out.close();
        return false;
    }
    out.close();
#endif
    return true;
}

unsigned int Fun(int64_t x) {
    unsigned int y = 0; // [0, 1024)
    while (x != 0) {
        y |= 1u << unsigned(x % 10);
        x /= 10;
    }
    return y;
}

void ComputeThread(int n, int64_t from, int64_t to, std::list<int64_t>
&numbers) {
    for (int64_t i = from; i <= to; ++i) {
        if (Fun(i) == Fun(i * n)) {
            mut.lock();
            numbers.push_back(i);
            mut.unlock();
        }
    }
}

void Compute(int n, int64_t thread_number, std::list<int64_t> &numbers) {
    int64_t loop_size = (MAX_VALUE - MIN_VALUE + 1) / thread_number;
    std::vector<std::thread> thr(thread_number);
    for (int64_t i = 0; i < thread_number; ++i) {
        if (i != thread_number - 1) {
            thr[i] = std::thread(ComputeThread, n, MIN_VALUE + loop_size * i,
MIN_VALUE + loop_size * (i + 1) - 1,
                                std::ref(numbers));
        } else {
            thr[i] = std::thread(ComputeThread, n, MIN_VALUE + loop_size * i,
MAX_VALUE, std::ref(numbers));
        }
    }
    for (std::thread &Thread : thr) {
        Thread.join();
    }
}

void Print(int n, int threads_num, const std::string &file_name, const
std::list<int64_t> &numbers) {
#ifdef FILE_OUT
    std::ofstream out(file_name);
    out << "Number n: " << n << '\n';
    out << "Amount of threads: " << threads_num << '\n';
    for (int64_t x : numbers) {
        out << x << '\n';
    }
    out.close();
#else
    for (int64_t x : numbers) {
        std::cout << x << '\n';
    }
#endif
}

```

```
int main() {
    std::ios_base::sync_with_stdio(false);
    int n, thread_number;
    std::string file_name;
    if (!Read(&n, &thread_number, &file_name)) {
        return 1;
    }

    std::list<int64_t> numbers;
    auto begin = std::chrono::steady_clock::now();

    Compute(n, thread_number, numbers);

    auto end = std::chrono::steady_clock::now();
    auto elapsed_ms =
std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);

    Print(n, thread_number, file_name, numbers);
    std::cout << "Time (computation): " << elapsed_ms.count() / 1000. << "
sec\n";
    return 0;
}
```