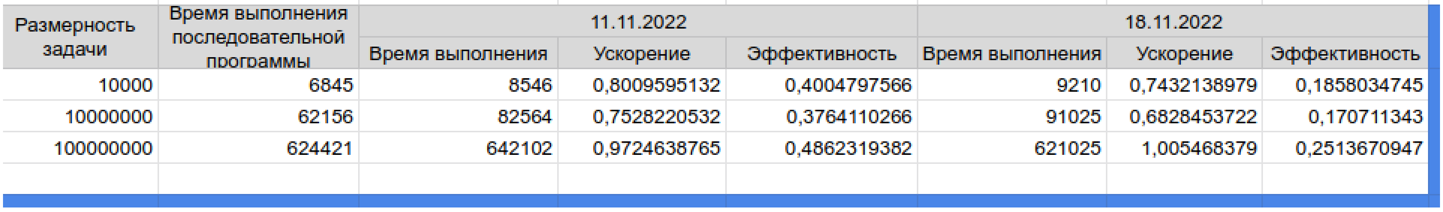
Лекция 18.11.2022

```
#include <queue>
#include <memory>
#include <mutex>
#include <condition_variable>
#include <iostream>
#include <fstream>

template<typename T>
class threadsafe_queue
{
private:
    mutable std::mutex mut;
    std::queue<T> data_queue;
    std::condition_variable data_cond;
public:
    threadsafe_queue()= default;

    threadsafe_queue(threadsafe_queue const& other) {
        std::lock_guard<std::mutex> lk(other.mut);
        data_queue = other.data_queue;
    }

    void push(T new_value) {
        std::lock_guard<std::mutex> lk(mut);
        data_queue.push(new_value);
        data_cond.notify_one();
    }

    void wait_and_pop(T& value) {
        std::unique_lock<std::mutex> lk(mut);
        data_cond.wait(lk, [this]{ return !data_queue.empty();});
        value = data_queue.pop();
    }
    std::shared_ptr<T> wait_and_pop(){
        std::unique_lock<std::mutex> lk(mut);
        data_cond.wait(lk,[this]{ return !data_queue.empty();});
        std::shared_ptr<T> res(std::make_shared<T>(data_queue.front()));
        data_queue.pop();
        return res;
    }

    bool try_pop(T& value){
        std::lock_guard<std::mutex> lk(mut);
        if(data_queue.empty()){
            return false;
        }
        value=data_queue.front();
        data_queue.pop();
        return true;
    }

    std::shared_ptr<T> try_pop(){
        std::lock_guard<std::mutex> lk(mut);
        if(data_queue.empty()){
            return std::shared_ptr<T>();
        }
        std::shared_ptr<T> res(std::make_shared<T>(data_queue.front()));
        data_queue.pop();
        return res;
    }
    bool empty() const{
        std::lock_guard<std::mutex> lk(mut);
        return data_queue.empty();
    }
};


using namespace std::chrono_literals;

class Consumer {
private:
    static std::ofstream fout;
public:
    void operator()(threadsafe_queue<std::string>& request_queue, std::mutex& mutex, bool& finished) {
        while (!finished) {
            fout << request_queue.try_pop() << " ";
        }
    }
};

class Producer {
private:
    static std::ifstream fin;
    std::string buffer_;
public:
    void operator()(threadsafe_queue<std::string>& request_queue, std::mutex& mutex, bool& finished) {
        while(!fin.eof()){
            {
                fin >> buffer_;
                request_queue.push(buffer_);
            }
        }
        {
            finished = true;
        }
    }
};

std::ifstream Producer::fin = std::ifstream("input.txt");
std::ofstream Consumer::fout = std::ofstream("output.txt");

int main() {
    auto start = std::chrono::high_resolution_clock::now();
    std::mutex mut;
    threadsafe_queue<std::string> request_queue;
    bool finished = false;
    std::thread t1(Consumer(), std::ref(request_queue), std::ref(mut), std::ref(finished));
    std::thread t2(Producer(), std::ref(request_queue), std::ref(mut), std::ref(finished));
    t1.join();
    t2.join();
    std::cout << "finished, time is " <<
std::chrono::duration_cast<std::chrono::microseconds>(std::chrono::high_resolution_clock::now()-
start).count() << " mcs\n";
}
```

Результаты вычислений:



Размерность задачи	Время выполнения последовательной программы	11.11.2022			18.11.2022		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
10000	6845	8546	0,8009595132	0,4004797566	9210	0,7432138979	0,1858034745
10000000	62156	82564	0,7528220532	0,3764110266	91025	0,6828453722	0,170711343
100000000	624421	642102	0,9724638765	0,4862319382	621025	1,005468379	0,2513670947