Козунов Алексей, 12 группа

Отчет

Код программы:

```
#include <iostream>
#include <vector>
#include <random>
#include <algorithm>
#include <execution>
#include <future>
#include <chrono>
#include <ctime>

using namespace std;
using namespace std::chrono;
int SIZE = 0;

void sizeEx(size_t size) {
	SIZE = size;
}

template <typename RAIter>
int find_max(RAIter beg, RAIter end)
{
	typename RAIter::difference_type len = end - beg;
	if (len < 3)
	{
		auto it = max_element(beg, end);
		return *it;
	}
	RAIter mid = beg + len / 2;
	auto handle = std::async(std::launch::async, find_max<RAIter>, mid, end);
	int maxVal = find_max(beg, mid);
	return max(maxVal, handle.get());
}

template<typename Iterator, typename T>
struct accumulate_block
{
	void operator()(Iterator first, Iterator last, T& result)
	{
		result = *std::max_element(first, last);
	}
};
template<typename Iterator, typename T>
T parallel(Iterator first, Iterator last, T init)
{
	unsigned long const length = std::distance(first, last);
	if (!length)
		return init;
	unsigned long const min_per_thread = 25;
	unsigned long const max_threads = (length + min_per_thread - 1) / min_per_thread;
	unsigned long const hardware_threads = std::thread::hardware_concurrency();
	unsigned long const num_threads = std::min(hardware_threads != 0 ? 
hardware_threads : 2,
		max_threads);
	unsigned long const block_size = length / num_threads;
	std::vector<T> results(num_threads);

	std::vector<std::thread> threads(num_threads - 1);
	Iterator block_start = first;
	for (auto i = 0; i < num_threads - 1; ++i) {
```

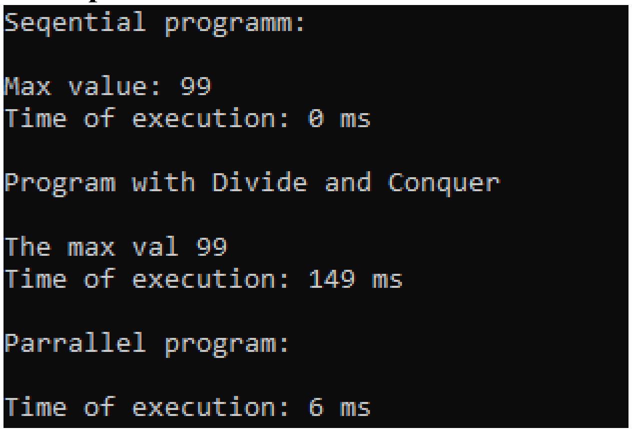
Козунов Алексей, 12 группа

```
		Iterator block_end = block_start;
		std::advance(block_end, block_size);
		threads[i] = std::thread(accumulate_block<Iterator, T>(), block_start, 
block_end,

			std::ref(results[i]));

		block_start = block_end;
	}
	accumulate_block<Iterator, T>() (block_start, last, results[num_threads - 1]);
	for (auto& entry : threads)
		entry.join();
	return *std::max_element(results.begin(), results.end());
}
void fill_row(std::vector<int>& row)
{
	srand(static_cast<unsigned>(time(0)));
	std::generate(row.begin(), row.end(), []() { return rand() % 1000; });
}

void main()
{
	sizeEx(1500);

	srand(time(NULL));
	int n = SIZE;
	vector<int> mat(n);
	cout << "Seqential programm: \n\n";
	for (int i = 0; i < n; i++)
	{
		mat[i] = rand() % 200 - 100;
	}
	int max = mat[0];
	auto start1 = chrono::high_resolution_clock::now();
	for (auto val : mat)
	{
		if (max < val)
			max = val;

	}

	auto end1 = chrono::high_resolution_clock::now();

	cout << "Max value: " << max << endl;
	cout << "Time of execution: " << chrono::duration_cast<chrono::milliseconds>(end1
-
		start1).count() << " ms" << endl;


	srand(time(NULL));
	cout << "\nProgram with Divide and Conquer\n\n";
	n = SIZE;
	std::vector<int> v(n);
	for (int i = 0; i < v.size(); i++)
	{
		v[i] = rand() % 200 - 100;
	}
	start1 = chrono::high_resolution_clock::now();
	cout << "The max val " << find_max(v.begin(), v.end()) << '\n';
	end1 = chrono::high_resolution_clock::now();
	cout << "Time of execution: " << chrono::duration_cast<chrono::milliseconds>(end1
-
		start1).count() << " ms" << endl;
```

```
    std::cout << "\nParrallel program:\n\n";
    const int N = SIZE;
    std::vector<int> matrix(N);
    fill_row(matrix);
    auto start = high_resolution_clock::now();
    int max_el = parallel(matrix.begin(), matrix.end(), 0);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop - start);
    std::cout << "Time of execution: " << duration.count() << " ms" << "\n";
    system("pause");
}
```

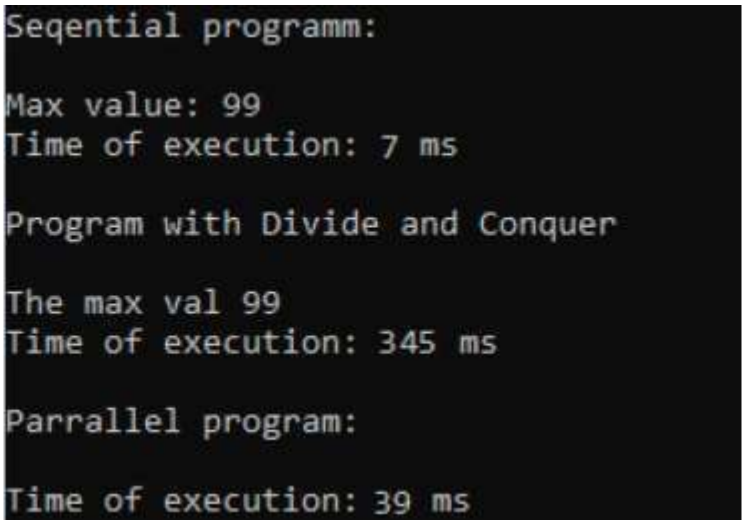
Результат:

- **Размер 1500:**



```
Seqential programm:

Max value: 99
Time of execution: 0 ms

Program with Divide and Conquer

The max val 99
Time of execution: 149 ms

Parrallel program:

Time of execution: 6 ms
```

- **Размер 150000:**



```
Seqential programm:

Max value: 99
Time of execution: 7 ms

Program with Divide and Conquer

The max val 99
Time of execution: 345 ms

Parrallel program:

Time of execution: 39 ms
```