Текст программы (три реализации):

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#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 < SIZE/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) {
        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; });
}


int main()
{
    sizeEx(10000);


    srand(time(NULL));
    int n = SIZE;
    vector<int> mat(n);
    cout << "Sequence: \n";
    for (int i = 0; i < n; i++)
    {
        mat[i] = rand() % 200 - 100;
    }
    int max = mat[0];


    auto start = std::chrono::steady_clock::now();
    for (auto val : mat)
    {
        if (max < val)
            max = val;
    }


    auto end = std::chrono::steady_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;


    cout << "Maximal value: " << max << endl;
    cout << "Time: " << elapsed_seconds.count() << " s." << endl;

    srand(time(NULL));
    cout << "Divide and Conquer\n";
    n = SIZE;
    std::vector<int> v(n);
    for (int i = 0; i < v.size(); i++)
    {
        v[i] = rand() % 200 - 100;
    }
    start = std::chrono::steady_clock::now();
    cout << "Maximal value: " << find_max(v.begin(), v.end()) << '\n';
    end = std::chrono::steady_clock::now();
    cout << "Time: " << std::chrono::duration<double>(end - start).count() << " s." << endl;
```

```
    std::cout << "Parrallel:\n";
    const int N = SIZE;
    std::vector<int> matrix(N);
    fill_row(matrix);
    start = std::chrono::steady_clock::now();
    int max_el = parallel(matrix.begin(), matrix.end(), 0);
    end = std::chrono::steady_clock::now();
    std::cout << "Time: " << std::chrono::duration<double>(end - start).count() << " s." << "\n";
    return 0;
}
```

Результаты экспериментов:

Размерность задачи	Последовательная программа, мкс	Параллельная программа			Разделяй и властвуй		
		Время выполнения	Ускорение	Эффективность	Время выполнения	Ускорение	Эффективность
10^4	305,6	1235,4	0,25632	0,06352	1256,3	0,21256	0,05325
10^5	3256,2	2012,5	1,4520	0,39525	1834,2	1,56325	0,39533
10^6	30523,7	14256,3	2,4023	0,60232	14921,2	2,00215	0,49652