ЗАДАНИЕ 03.12.2021.

Поиск значения максимального элемента вектора

Выполнил:
студент 3 курса 13 группы кафедры ТП.

Петров Андрей Александрович

Реализуем последовательную программу для поиска максимального элемента вектора

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <thread>
#include <algorithm>

using namespace std;

const int n = 1000000;
vector<int> arr(n);

void fillVector() {
    srand(time(0));
    for (auto &item: arr) {
        item = rand();
    }
}

int main() {
    fillVector();

    double start_time = clock();
    std::sort(arr.begin(), arr.end());
    int maxItem = arr.back();

    double end_time = clock();
    double exec_time = (end_time - start_time) / CLOCKS_PER_SEC;
    cout << "Total time: " << exec_time << "s" << "\n";
    printf("Max item: %d", maxItem);
    return 0;
}
```

Реализуем параллельную программу для поиска максимального элемента вектора на основе параллельной версии std::accumulate [Энтони Уильямс, 2.4.]

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <thread>
```

```cpp
#include <algorithm>
using namespace std;

const int n = 1000000;
vector<int> vec(n);

void fillVector() {
    srand(time(0));
    for (auto &item: vec) {
        item = rand();
    }
}

template<typename Iterator, typename T>
struct max_item_block {
    void operator()(Iterator first, Iterator last, T &result) {
        result = *max_element(first, last);
    }
};

template<typename Iterator, typename T>
T parallel_vec_max_item(Iterator first, Iterator last, T init) {
    unsigned long const length = 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 = thread::hardware_concurrency();
    unsigned long const num_threads = min(hardware_threads != 0 ?
                                      hardware_threads : 2, max_threads);
    unsigned long const block_size = length / num_threads;
    vector<T> results(num_threads);
    vector<thread> threads(num_threads - 1);
    Iterator block_start = first;
    for (auto i = 0; i < num_threads - 1; ++i) {
        Iterator block_end = block_start;
        advance(block_end, block_size);
        threads[i] = thread(max_item_block<Iterator, T>(), block_start,
                                            block_end, ref(results[i]));
        block_start = block_end;
    }
    max_item_block<Iterator, T>()(block_start, last, results[num_threads -
                                                                1]);
    for (auto &entry: threads)
        entry.join();
    return *max_element(results.begin(), results.end());
}

int main() {
    fillVector();

    double start_time = clock();
    auto maxItem = parallel_vec_max_item(vec.begin(), vec.end(), 0);
    double end_time = clock();
```

```
    double exec_time = (end_time - start_time) / CLOCKS_PER_SEC;
    cout << "Total time: " << exec_time << "s" << "\n";
    printf("Max item: %d", maxItem);
    return 0;
}
```

Реализуем параллельную программу для поиска максимального элемента вектора по принципу Divide and Conquer

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <thread>
#include <algorithm>
#include <future>

using namespace std;

const int n = 100000;
const int max_range_length = 1000;
vector<int> vec(n);

void fillVector() {
    srand(time(0));
    for (auto &item: vec) {
        item = rand();
    }
}

template<typename Iterator>
int parallel_vec_max_item(Iterator first, Iterator last) {
    ptrdiff_t const range_length = last - first;
    if (!range_length) return numeric_limits<int>::min();
    if (range_length <= max_range_length) return *max_element(first, last);

    Iterator const middle = first + (range_length / 2);

    future<int> task = async(&parallel_vec_max_item <Iterator>, first,
middle);
    int maxItem1 = 0;
    int maxItem2 = 0;
    try {
        maxItem1 = parallel_vec_max_item(middle, last);
    }
    catch (...) {
        task.wait();
        throw;
    }

    maxItem2 = task.get();
    return max(maxItem1, maxItem2);
}
```

```
int main() {
    fillVector();

    double start_time = clock();
    auto maxItem = parallel_vec_max_item(vec.begin(), vec.end());
    double end_time = clock();

    double exec_time = (end_time - start_time) / CLOCKS_PER_SEC;
    cout << "Total time: " << exec_time << "s" << "\n";
    printf("Max item: %d", maxItem);
    return 0;
}
```

Произведем вычислительные эксперименты.

Размер n	Последовательная программа, с.	Параллельная программа, с.					
		Версия Энтони Уильямса			Divide and Conquer		
		Время, с.	Ускорение	Эффективность	Время, с.	Ускорение	Эффективность
100 000	0,043	0,001	43	21,5	0,009	43	21,5
1 000 000	0,348	0,003	116	58	0,131	116	58
10 000 000	3,41	0,016	213,125	106,5625			