**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>**#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 |  |  |  |