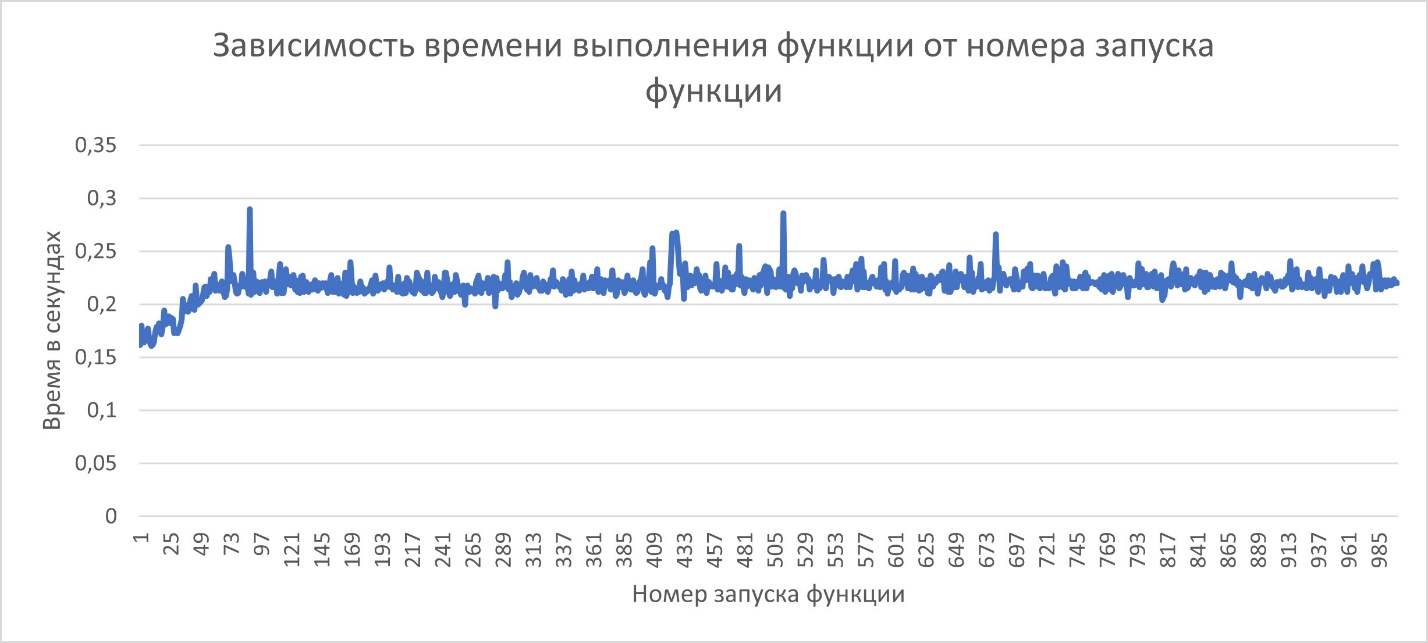
*На защиту: выполните не менее 100 итераций алгоритма (чем больше, тем лучше получатся результаты) на одних и тех же входных данных используя все 12 ядер. Оцените (1) меняется ли результат от первых итераций к последним (2) какова погрешность измерения и какой характер она имеет?*

Для данного задания в массиве размером в элементов был выполнен поиск максимального элемента раз.

1. *Время выполнения:*

Первые запуски занимали секунд, последние – секунды.



Увеличение времени выполнения может быть связано со следующими причинами:

1. В современных процессорах допускается увеличенный теплопакет/повышенные частоты на кратковременные затратные операции, но после пары десятков секунд они ограничиваются, замедляя выполнение функции.
2. ОС также может выдавать больший приоритет для кратковременных операций, после чего планировщик изменяет распределение ресурсов между всеми процессами.
3. При первых итерациях цикла данные могут попадать в кэш, что ускоряет вычисления, но в дальнейшем либо кэш кончается, либо опять же система ограничивает его использование.
4. *Погрешности:*
5. Используемая для замеров времени функция omp\_get\_wtime() может быть немного неточна - случайная погрешность.
6. Другие процессы влияют на ресурсы – случайная погрешность.

Троттлинга во время выполнения зафиксировано не было, поэтому его в причины не записываю.

*Приложение:*

1. Исходный код используемой программы:

#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>   
  
void initialize\_array(const int random\_seed, int \*array, const int count){  
 srand(random\_seed);  
  
 for (int i = 0; i < count; i++){  
 array[i] = rand();  
 }  
}  
  
double find\_max\_element(const int \*array, const int threads, const int count){  
 int max = -1;  
 int num;  
 double time\_spent;  
  
#pragma omp parallel num\_threads(threads) shared(array, count, time\_spent) private(num) reduction(max: max) default(none)  
 {  
 num = omp\_get\_thread\_num();  
 // printf("%d\n", num);  
 double begin = omp\_get\_wtime();  
#pragma omp for  
 for (int i = 0; i < count; i++){  
 if (array[i] > max){  
 max = array[i];  
 }  
 }  
 double end = omp\_get\_wtime();  
 time\_spent = end - begin;  
 // printf("-- My lmax is: %d;\n", max);  
 }  
  
 // printf("Max is: %d;\n", max);  
 return time\_spent;  
}  
  
int main(int argc, char \*\*argv){  
 const int count = 1000000000; ///< Number of array elements  
 const int random\_seed = 920215; ///< RNG seed  
 double time\_spent = 0;  
 const int threads = 12;  
 const int runs\_num = 1000;  
  
 int \*array = malloc(count \* sizeof(int)); ///< The array we need to find the max in  
  
 /\* Determine the OpenMP support \*/  
 printf("OpenMP: %d;\n======\n", \_OPENMP);  
  
 initialize\_array(random\_seed, array, count);  
  
 for (int i = 0; i < runs\_num; i++){  
 find\_max\_element(array, threads, count);  
 time\_spent = find\_max\_element(array, threads, count);  
 printf("Run N%d\nTime spent: %.14lf\n", i, time\_spent);  
 }  
 return (0);  
}