Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

**Лабораторная работа №1:**

**«Введение в параллельные вычисления. Технология OpenMP»**

Антон Гатченко Б22-525

2024 г.

*Используемая рабочая среда:*

* Процессор - AMD Ryzen 5 5600H (laptop), 6c/12t
* Оперативная память – DDR4 16 ГБ
* ОС - Windows 10 Pro 22H2 19045.4780, 64 bit
* IDE - CLion 2024.1.1, GCC 13.1, OpenMP 201511

*Анализ приведенного алгоритма*:

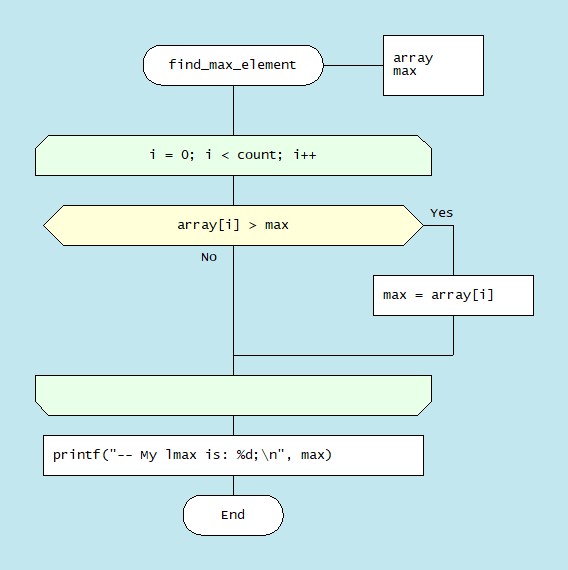
Задаем переменную max неким числом, заведомо меньшим чем хотя бы один элемент массива. Проходим весь массив, сравнивая его элементы с max. При обнаружении элемента со значением большим, чем у max, присваиваем его значение переменной max.

Теоретическая временная сложность:

Теоретическое ускорение параллельных вычислений:

Теоретическая эффективность параллельных вычислений:

Блок-схема алгоритма:



*Директивы OpenMP в функции find\_max\_element:*

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);  
 clock\_t begin = clock();  
#pragma omp for  
 for (int i = 0; i < count; i++){  
 if (array[i] > max){  
 max = array[i];  
 }  
 }  
 clock\_t end = clock();  
 time\_spent = (double) (end - begin) / CLOCKS\_PER\_SEC;  
 printf("-- My lmax is: %d;\n", max);  
 }  
  
 printf("Max is: %d;\n", max);  
 return time\_spent;  
}

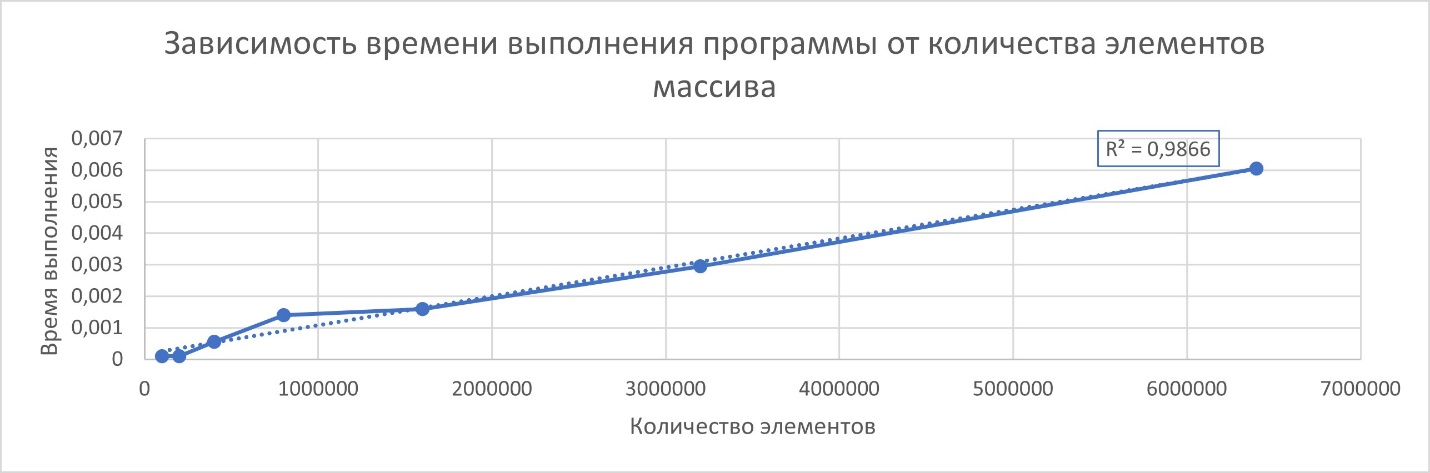
Директива parallel – создание потоков/нитей. Блок (в фигурных скобках) после данной директивы будет распараллелен. В данной функции действует от num = omp\_get\_thread\_num() до printf("-- My lmax is: %d;\n", max).

Директива for - явное распараллеливание цикла for, находящегося сразу после директивы, при этом каждый поток начнет со своего индекса. Без этой директивы каждый поток будет идти от начала и до конца цикла, проходя все индексы. В данной функции действует на цикл for (int i = 0; i < count; i++){…}.

*Экспериментальное вычисление временной сложности алгоритма:*

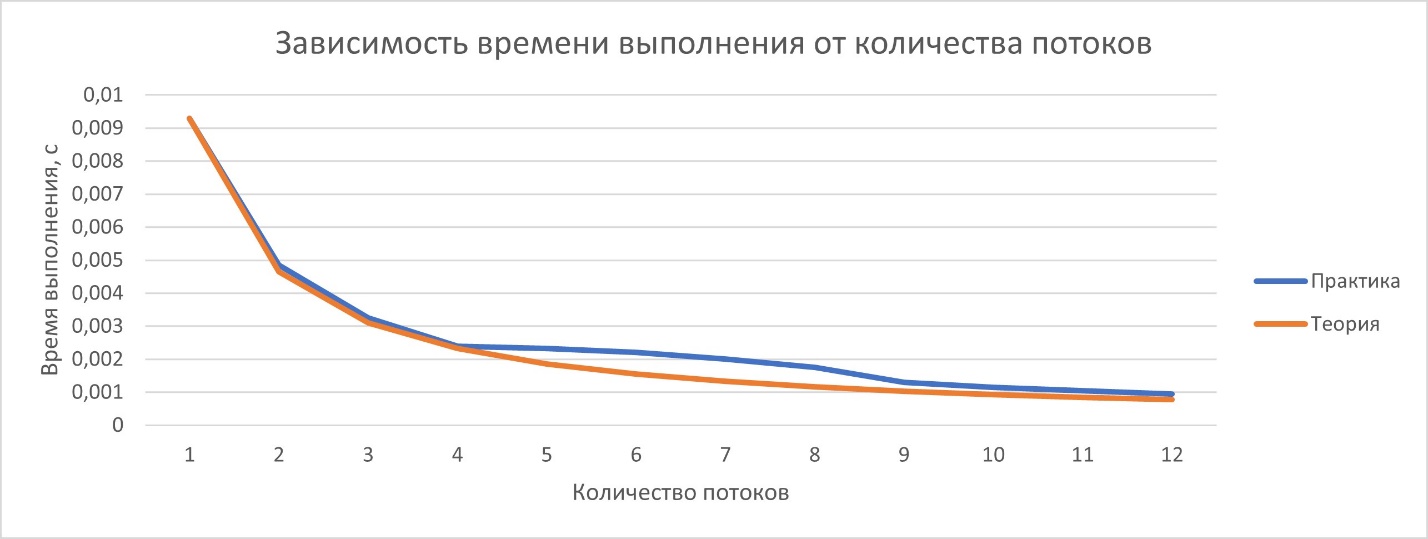
Графики:

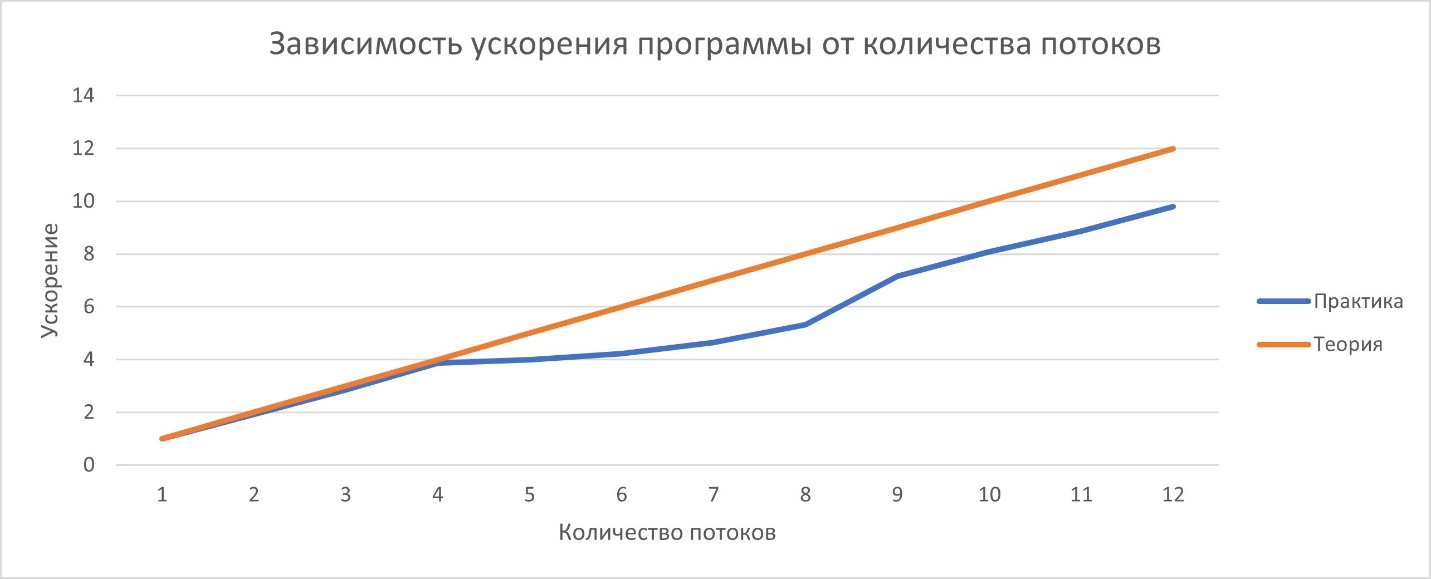


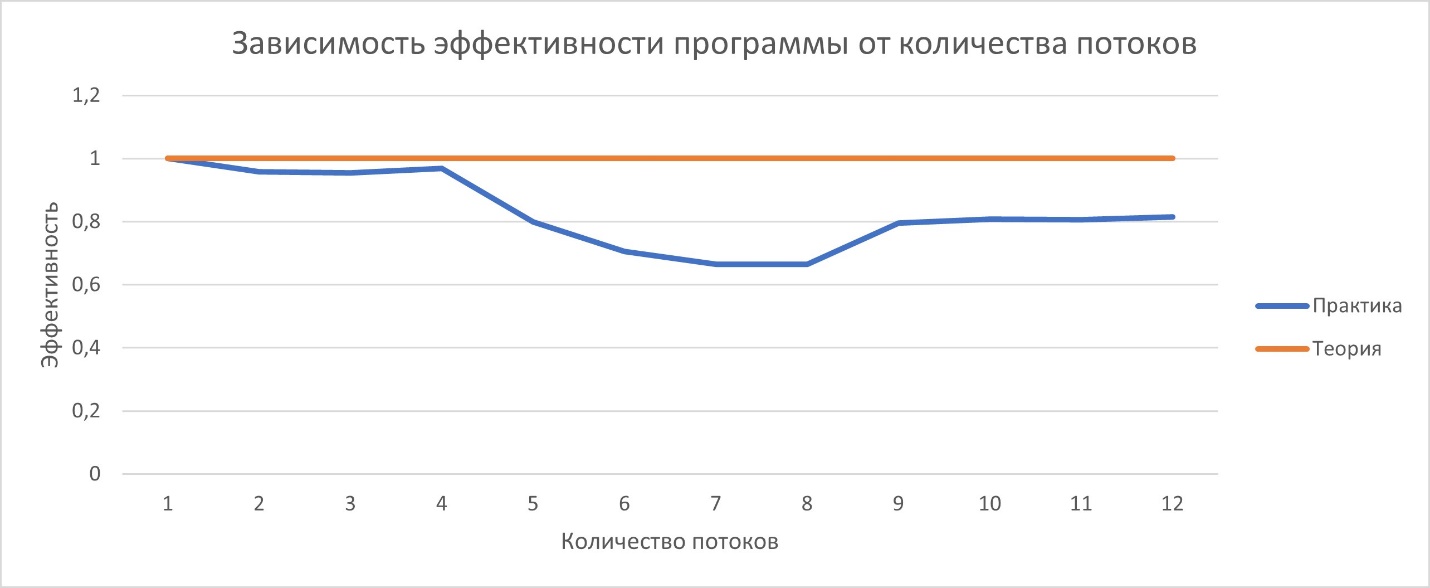


Полученные экспериментальные значения соответствуют теоретической сложности алгоритма в .

*Графики выполнения программы при различном количестве потоков:*







*Заключение:*

В ходе данной лабораторной работы был изучен процесс использования технологии OpenMP для организации параллельных вычислений на примере функции поиска максимального элемента в массиве. Программа была запущена при разном количестве потоков, что позволило оценить влияние распараллеливания на производительность.

Результаты тестов показали, что теоретическая и практическая временная сложность алгоритма сходятся и соответствуют , а также наблюдается ожидаемое ускорение выполнения программы при увеличении числа потоков, также близкое к теоретическим предположениям.

Эффективность параллелизации начинает снижаться при большом количестве потоков, что может объясняться затратами на управление синхронизацией потоков.

*Приложение:*

1. Исходный код программы с измерением времени работы программы:

#include <stdio.h>  
#include <stdlib.h>  
#include <omp.h>  
#include <time.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; ///< The maximal element  
 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);  
 clock\_t begin = clock();  
#pragma omp for  
 for (int i = 0; i < count; i++){  
 if (array[i] > max){  
 max = array[i];  
 }  
 }  
 clock\_t end = clock();  
 time\_spent = (double) (end - begin) / CLOCKS\_PER\_SEC;  
// 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 = 10000000; ///< Number of array elements  
 int random\_seed = 920215; ///< RNG seed  
 double time\_spent = 0;  
  
 int \*array = (int \*) malloc(count \* sizeof(int)); ///< The array we need to find the max in  
  
 /\* Determine the OpenMP support \*/  
 printf("OpenMP: %d;\n======\n", \_OPENMP);  
 const int runs\_num = 20;  
  
 for (int t = 1; t <= 12; t++){  
 for (int i = 0; i < runs\_num; i++){  
 initialize\_array(random\_seed, array, count);  
 random\_seed += 1234;  
 time\_spent += find\_max\_element(array, t, count);  
 }  
 printf("Threads: %d\nSeconds spent: %lf", t, time\_spent / runs\_num);  
 time\_spent = 0;  
 random\_seed = 920215;  
 }  
 return (0);  
}

1. Таблица времени работы программы

|  |  |  |
| --- | --- | --- |
| Время работы программы | | |
| **Количество потоков** | **Практика** | **Теория** |
| 1 | 0,0093 | 0,0093 |
| 2 | 0,0048501 | 0,00465 |
| 3 | 0,0032501 | 0,0031 |
| 4 | 0,0024001 | 0,002325 |
| 5 | 0,0023301 | 0,00186 |
| 6 | 0,0022002 | 0,00155 |
| 7 | 0,0020005 | 0,001328571 |
| 8 | 0,0017501 | 0,0011625 |
| 9 | 0,0013001 | 0,001033333 |
| 10 | 0,0011503 | 0,00093 |
| 11 | 0,00105 | 0,000845455 |
| 12 | 0,0009504 | 0,000775 |

1. Таблица ускорения программы

|  |  |  |
| --- | --- | --- |
| Ускорение программы | | |
| **Количество потоков** | **Практика** | **Теория** |
| 1 | 1 | 1 |
| 2 | 1,917486237 | 2 |
| 3 | 2,861450417 | 3 |
| 4 | 3,874838548 | 4 |
| 5 | 3,991245011 | 5 |
| 6 | 4,226888465 | 6 |
| 7 | 4,648837791 | 7 |
| 8 | 5,313982058 | 8 |
| 9 | 7,1532959 | 9 |
| 10 | 8,084847431 | 10 |
| 11 | 8,857142857 | 11 |
| 12 | 9,785353535 | 12 |

1. Таблица эффективности программы

|  |  |  |
| --- | --- | --- |
| Эффективность программы | | |
| **Количество потоков** | **Практика** | **Теория** |
| 1 | 1 | 1 |
| 2 | 0,958743119 | 1 |
| 3 | 0,953816806 | 1 |
| 4 | 0,968709637 | 1 |
| 5 | 0,798249002 | 1 |
| 6 | 0,704481411 | 1 |
| 7 | 0,664119684 | 1 |
| 8 | 0,664247757 | 1 |
| 9 | 0,794810656 | 1 |
| 10 | 0,808484743 | 1 |
| 11 | 0,805194805 | 1 |
| 12 | 0,815446128 | 1 |