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

Кафедра №42 «Криптология и кибербезопасность»

**Лабораторная работа №2:**

**«Выделение ресурса параллелизма. 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 2024.2.1, GCC 13.1, OpenMP 201511

*Анализ приведенного алгоритма*:

В заданном массиве проходим по элементам, в случае совпадения с искомым значением сохраняем индекс и прерываем цикл.

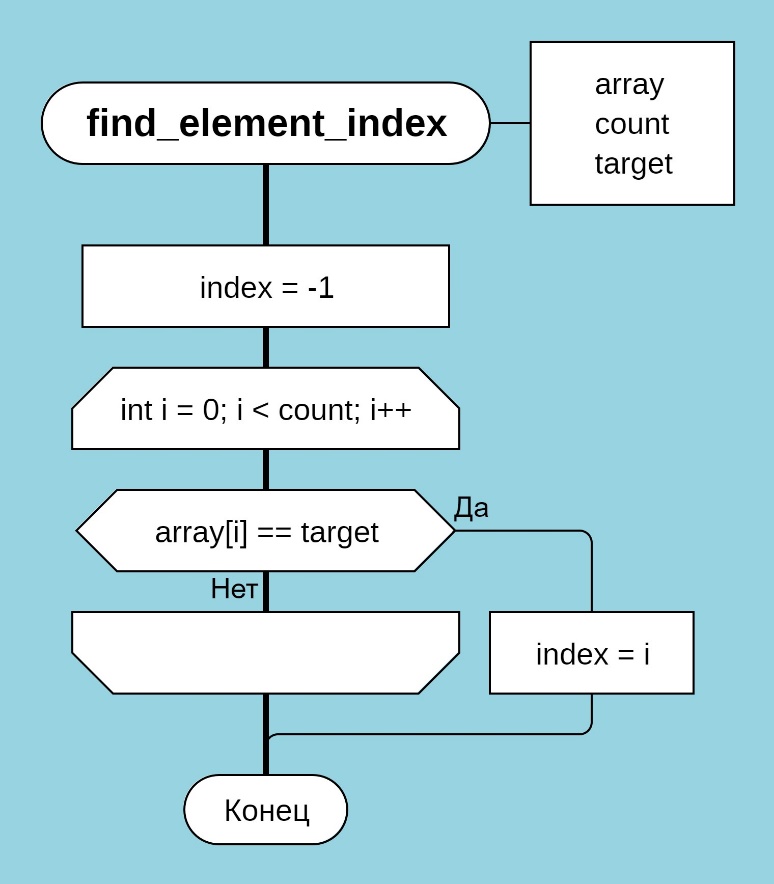
– количество элементов массива, – количество элементов, равных искомому.

Теоретическая временная сложность: наилучшая - ; наихудшая - ; средняя -

Теоретическое идеальное ускорение параллельных вычислений:

Теоретическая идеальная эффективность параллельных вычислений:

Блок-схема алгоритма:



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

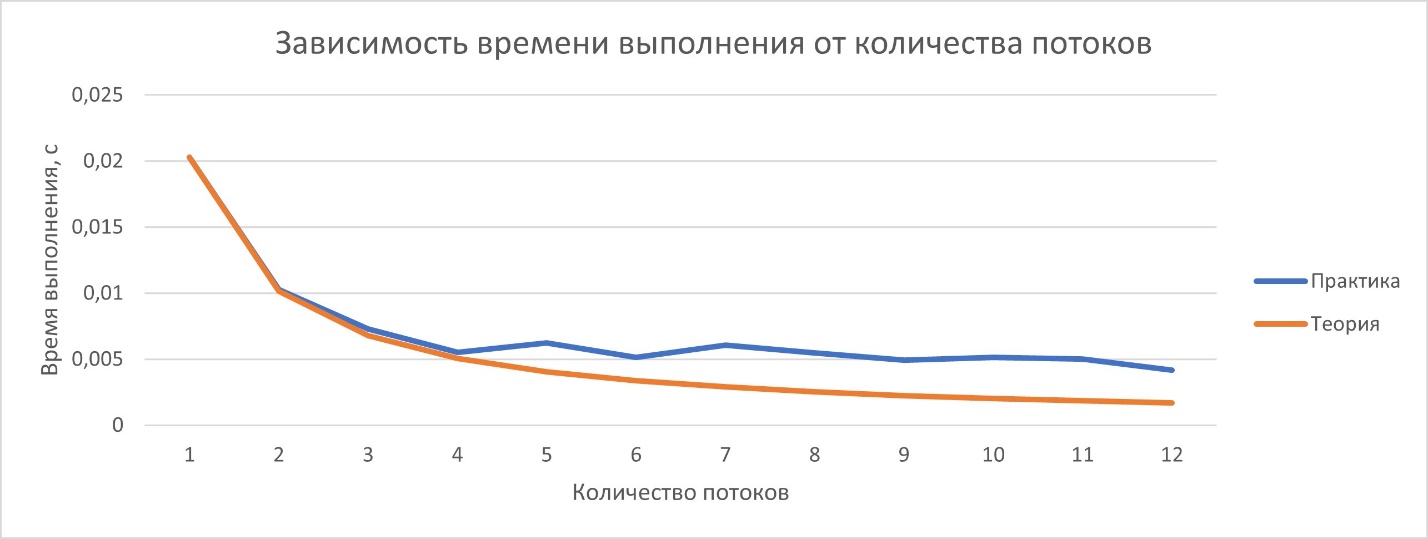
int find\_element\_index(const int \*array, const int count, const int target, const int threads){  
 int num;  
 int index = -1;  
 int elementIsFound = 0;  
  
#pragma omp parallel num\_threads(threads) shared(array, count, target, index, elementIsFound) private(num) default(none)  
 {  
 #pragma omp for  
 for (int i = 0; i < count; i++){  
 if (elementIsFound){  
 i = count;  
 continue;  
 }  
 if (array[i] == target){  
 index = i;  
 elementIsFound = 1;  
 }  
 }  
 }  
 return index;  
}

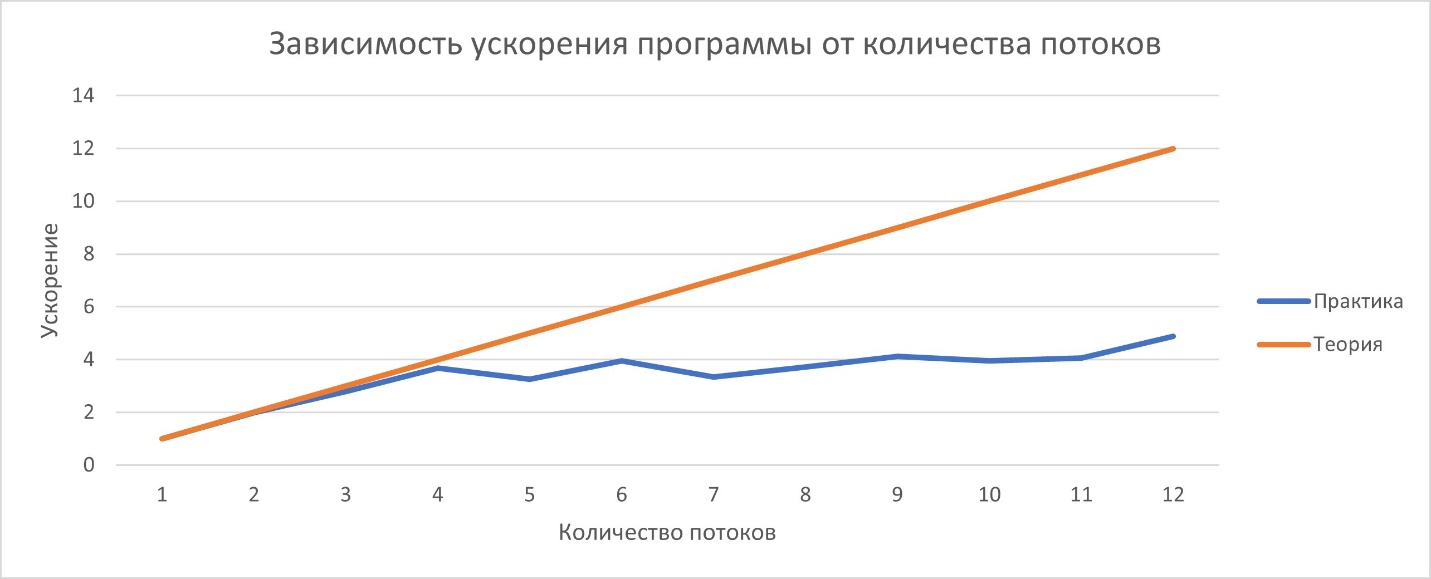
Директива parallel – создание потоков/нитей. Блок (в фигурных скобках) после данной директивы будет распараллелен.

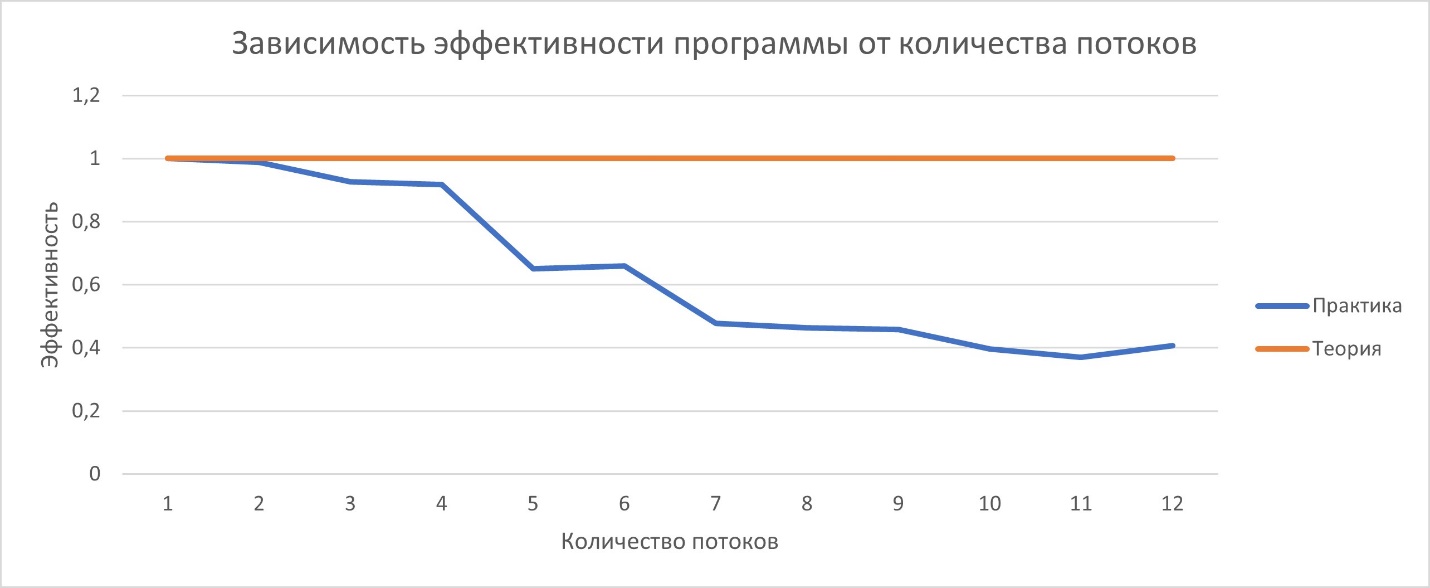
Директива for - явное распараллеливание цикла for, находящегося сразу после директивы, при этом каждый поток начнет со своего индекса. Без этой директивы каждый поток будет идти от начала и до конца цикла, проходя все индексы.

Так как data race в данном случае не влияет на корректность алгоритма (по индексу будет находиться элемент с нужным значением), то можно не использовать дополнительные директивы (critical, atomic write).

*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 элемента с заданным значением в массиве. Программа была запущена при разном количестве потоков, что позволило оценить влияние распараллеливания на производительность и время выполнения алгоритма.

Результаты тестов показали, что теоретические предположения об ускорении и эффективности выполнения программы при увеличении числа потоков расходятся с практическими. Оптимальное количество потоков для данного алгоритма – 4.

Эффективность параллелизации начинает резко снижаться при количестве потоков больше 4, что может объясняться затратами на управление синхронизацией переменных (флаг для остановки остальных потоков при нахождении индекса нужного элемента), из-за чего потоки останавливаются не в одно и то же время, сторонними процессами системы, а также влиянием планировщика ОС на выделяемые программе ресурсы.

*Приложение:*

1. Исходный код программы с измерением времени работы 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() \* rand();  
 }  
}  
  
double find\_element\_index(const int \*array, const int count, const int target, const int threads){  
 int num;  
 int index = -1;  
 int elementIsFound = 0;  
 double time\_spent = 0;  
  
#pragma omp parallel num\_threads(threads) shared(array, count, target, index, elementIsFound) private(num) reduction(max: time\_spent) default(none)  
 {  
 num = omp\_get\_thread\_num();  
 double begin = omp\_get\_wtime();  
 #pragma omp for  
 for (int i = 0; i < count; i++){  
 if (elementIsFound){  
 i = count;  
 continue;  
 }  
 if (array[i] == target){  
 index = i;  
 printf("%d\n", index);  
 elementIsFound = 1;  
 }  
 }  
 double end = omp\_get\_wtime();  
 time\_spent = end - begin;  
 }  
 return time\_spent;  
}  
  
int main(int argc, char \*\*argv){  
 const int count = 50000000; ///< Number of array elements  
 int random\_seed = 920214; ///< RNG seed  
 const int target = 32000; ///< Number to look for  
 const int threads = 12;  
 int \*array = malloc(count \* sizeof(int));  
 int index = -1; ///< The index of the element we need  
  
 const int runs\_num = 30;  
 double time\_spent = 0;  
  
 /\* Determine the OpenMP support \*/  
 printf("OpenMP: %d\n", \_OPENMP);  
  
 for (int t = 1; t <= threads; t++){  
 for (int i = 0; i < runs\_num; i++){  
 initialize\_array(random\_seed, array, count);  
 random\_seed += 12345;  
 time\_spent += find\_element\_index(array, count, target, t);  
 }  
 printf("%.14lf\n", time\_spent / runs\_num);  
 time\_spent = 0;  
 random\_seed = 920214;  
 }  
 return 0;  
}

1. Таблица времени работы программы

|  |  |  |
| --- | --- | --- |
| Время работы программы | | |
| **Количество потоков** | **Практика** | **Теория** |
| 1 | 0,020299983 | 0,020299983 |
| 2 | 0,010266654 | 0,010149992 |
| 3 | 0,007300003 | 0,006766661 |
| 4 | 0,005533322 | 0,005074996 |
| 5 | 0,006233327 | 0,004059997 |
| 6 | 0,005133319 | 0,003383331 |
| 7 | 0,00606668 | 0,002899998 |
| 8 | 0,005466636 | 0,002537498 |
| 9 | 0,004933349 | 0,002255554 |
| 10 | 0,005133319 | 0,002029998 |
| 11 | 0,005000027 | 0,001845453 |
| 12 | 0,004166651 | 0,001691665 |

1. Таблица ускорения программы

|  |  |  |
| --- | --- | --- |
| Ускорение программы | | |
| **Количество потоков** | **Практика** | **Теория** |
| 1 | 1 | 1 |
| 2 | 1,977273572 | 2 |
| 3 | 2,780818309 | 3 |
| 4 | 3,668679345 | 4 |
| 5 | 3,25668529 | 5 |
| 6 | 3,954553266 | 6 |
| 7 | 3,346143719 | 7 |
| 8 | 3,713432345 | 8 |
| 9 | 4,114848113 | 9 |
| 10 | 3,954553266 | 10 |
| 11 | 4,059974664 | 11 |
| 12 | 4,872014511 | 12 |

1. Таблица эффективности программы

|  |  |  |
| --- | --- | --- |
| Эффективность программы | | |
| **Количество потоков** | **Практика** | **Теория** |
| 1 | 1 | 1 |
| 2 | 0,988636786 | 1 |
| 3 | 0,926939436 | 1 |
| 4 | 0,917169836 | 1 |
| 5 | 0,651337058 | 1 |
| 6 | 0,659092211 | 1 |
| 7 | 0,478020531 | 1 |
| 8 | 0,464179043 | 1 |
| 9 | 0,457205346 | 1 |
| 10 | 0,395455327 | 1 |
| 11 | 0,369088606 | 1 |
| 12 | 0,406001209 | 1 |