Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №2: «Выделение ресурса параллелизма. Технология OpenMP»

Одношивкин Илья, Б21-515

Описание архитектуры:

```
[qcold@DESKTOP-J2NEN3H ~]$ lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         48 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  12
  On-line CPU(s) list:   0-11
[qcold@DESKTOP-J2NEN3H ~]$ sudo hostnamectl
     Static hostname: (unset)
  Transient hostname: DESKTOP-J2NEN3H
            Icon name: computer-laptop
              Chassis: laptop 💻
           Machine ID: caf94732efe24b519ce9ca85095c24f4
              Boot ID: b14557368dec4e46914ce5e5b4b4c546
    Operating System: Fedora Linux 38 (Workstation Edition)
         CPE OS Name: cpe:/o:fedoraproject:fedora:38
      OS Support End: Tue 2024-05-14
OS Support Remaining: 7month 1w 6d
              Kernel: Linux 6.5.5-200.fc38.x86_64
        Architecture: x86-64
     Hardware Vendor: HUAWEI
      Hardware Model: NBM-WXX9
    Firmware Version: 2.09
       Firmware Date: Wed 2022-03-23
                       misalignsse 3dnowprefetch osvw ibs skinit wd
                      t tce topoext perfctr_core perfctr_nb bpext p
                      erfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate
                       ssbd mba ibrs ibpb stibp vmmcall fsgsbase bm
                      i1 avx2 smep bmi2 cqm rdt_a rdseed adx smap c
                      lflushopt clwb sha_ni xsaveopt xsavec xgetbv1
                       cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_
                      local clzero irperf xsaveerptr rdpru wbnoinvd
                       cppc arat npt lbrv svm_lock nrip_save tsc_sc
                      ale vmcb_clean flushbyasid decodeassists paus
                      efilter pfthreshold avic v_vmsave_vmload vgif
                       v_spec_ctrl umip rdpid overflow_recov succor
                       smca
Virtualization features:
  Virtualization:        AMD-V
[qcold@DESKTOP-J2NEN3H ~]$ free
               total        used        free      shared  buff/cache
 available
Mem:         7428988     3465828      223860       49408     3739300
   3643816
Swap:        7428092     1941760     5486332
[qcold@DESKTOP-J2NEN3H ~]$
```

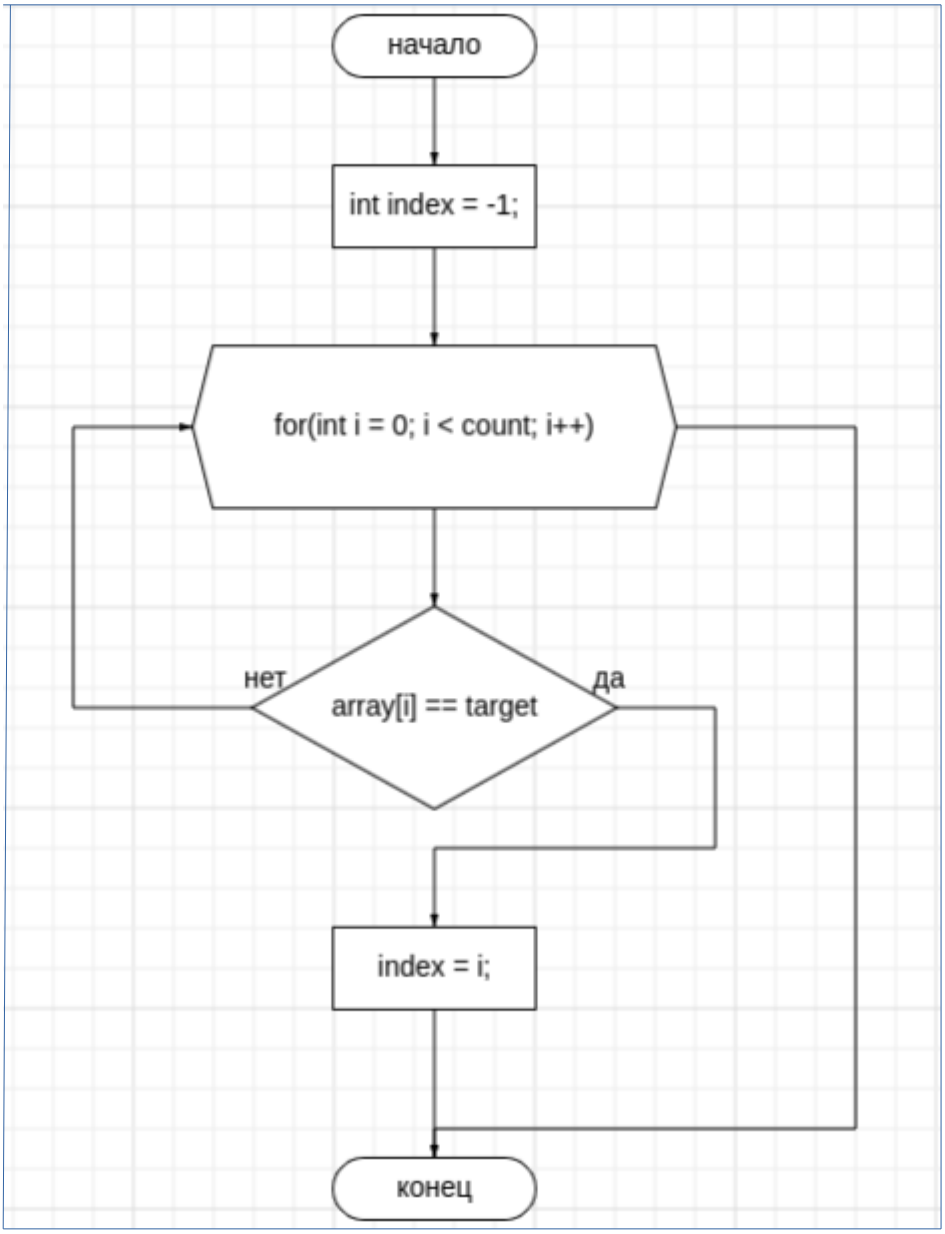
Среда разработки:

Компилятор: gcc (GCC) 13.2.1 20230728 (Red Hat 13.2.1-1)
Средство сборки: Makefile
Текстовый редактор: Visual Studio Code
Версия OPENMP: 201511

Описание алгоритма:

Работа алгоритма заключается в следующем: он просматривает каждый элемент массива последовательно. Если он находит элемент, который равен заданному значению, он сохраняет индекс этого элемента и завершает поиск. Если такой элемент не найден, индекс остается равным -1.

Блок-схема алгоритма:



Описание директив и функций OpenMP:

parallel - директива OpenMP, которая указывает начало блока параллельной части программы. Этот блок будет выполняться параллельно в нескольких потоках.

num_threads(int n) - директива OpenMP, которая указывает, что блок кода, следующий за директивой parallel, должен выполняться с использованием n потоков. В данном случае, threads переменная в цикле for задает количество потоков.

shared(array, count, target, threads, found) - директива, которая указывает, что указанные переменные (array, count, target, threads, found) являются общими для всех потоков и будут доступны каждому потоку внутри блока parallel.

reduction(max : index) - директива, которая указывает, что переменная index будет локальной для каждого потока, и по завершении блока parallel, ее значение будет сведено к максимальному значению среди всех потоков с использованием операции max.

default(none) - директива OpenMP, которая устанавливает строгий режим контроля переменных.

График №1:

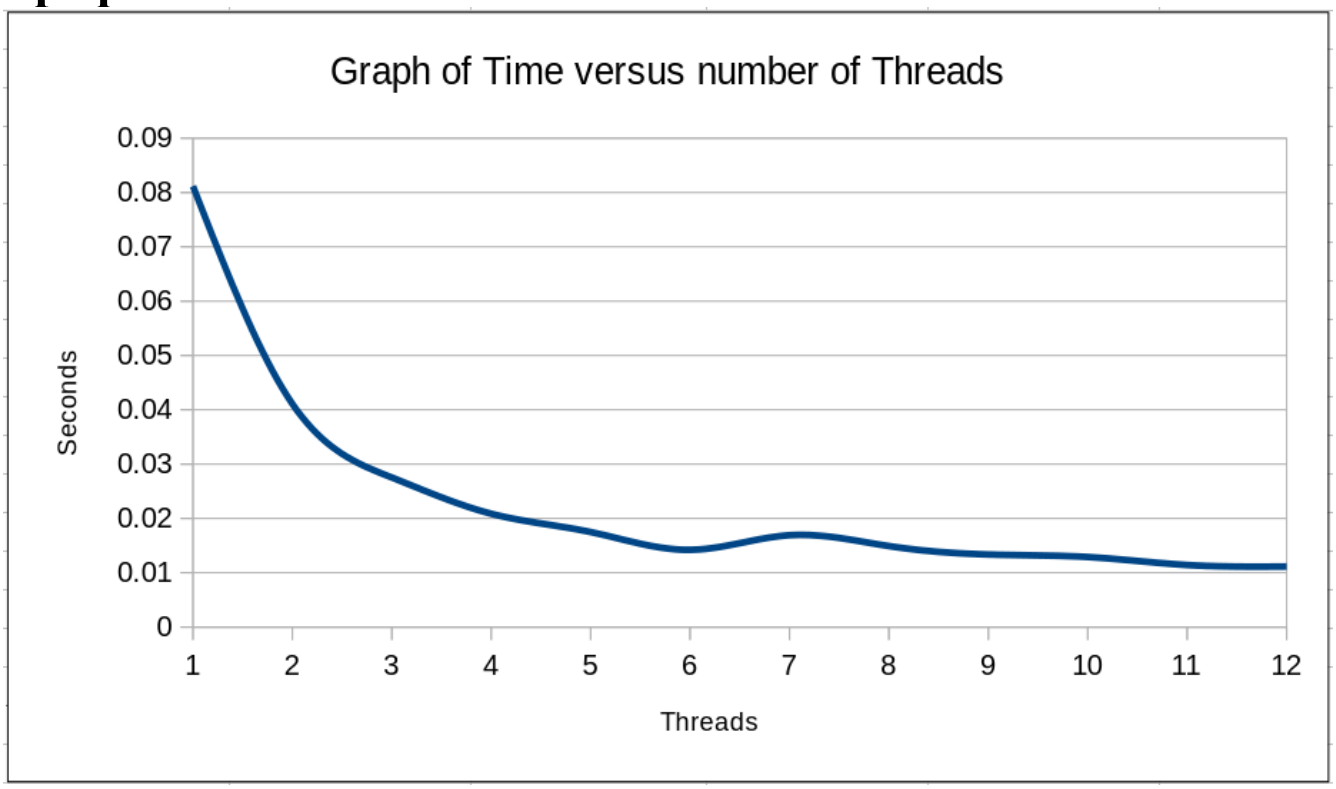


График №2:

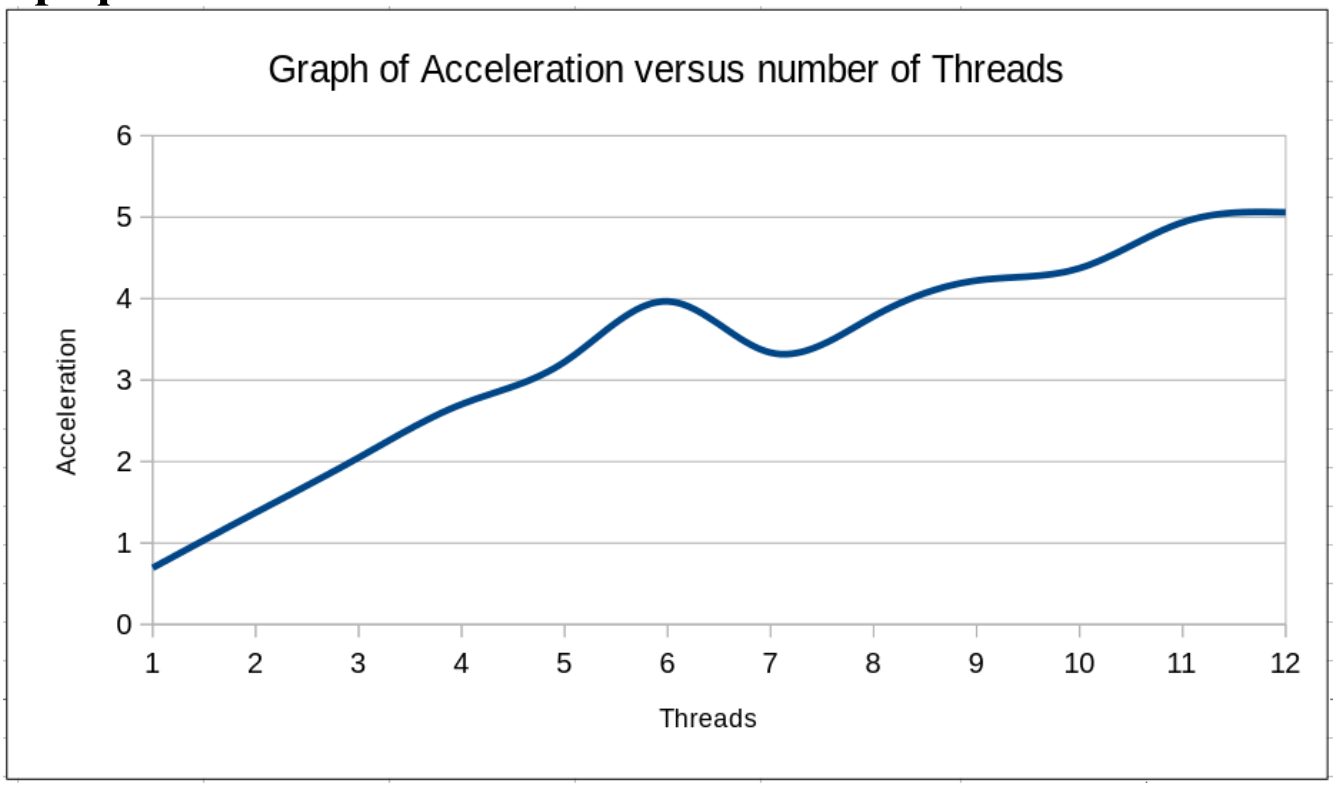


График №3:

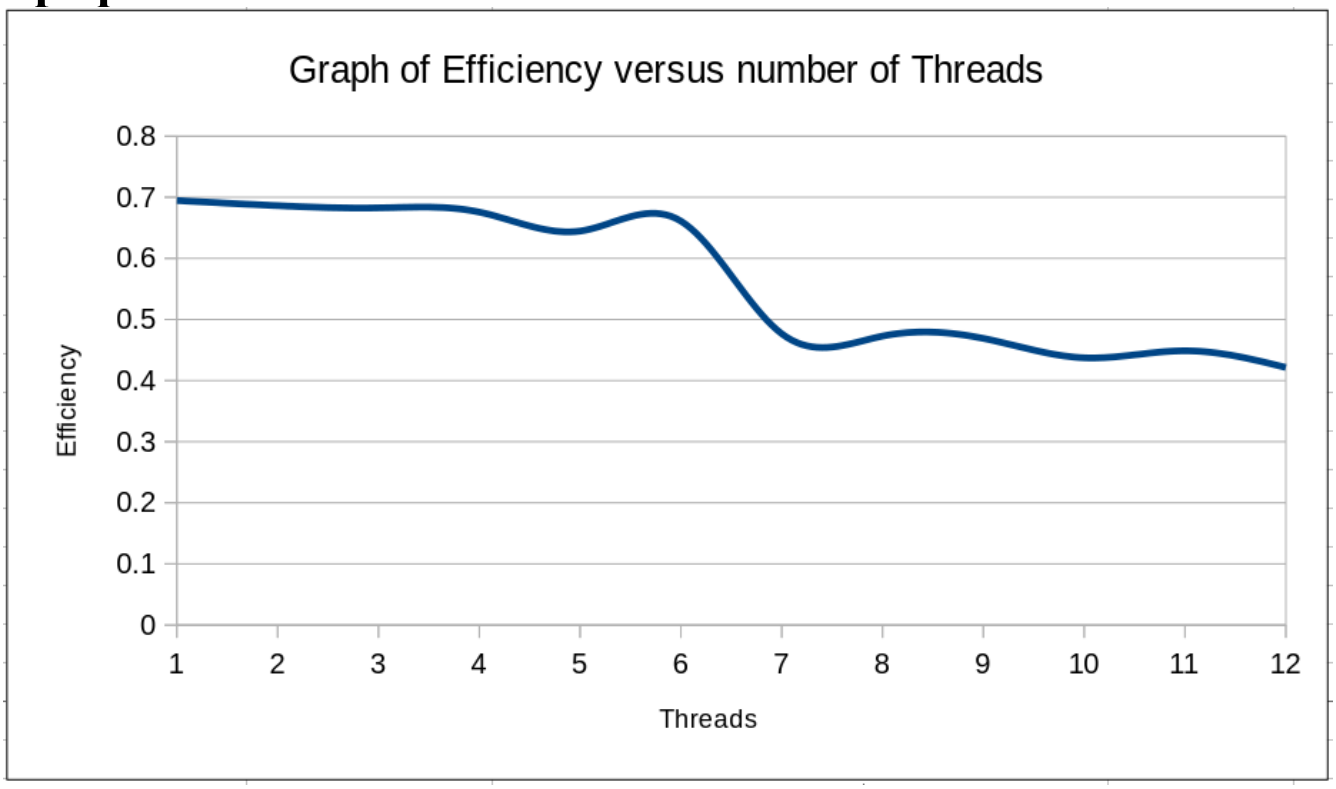


График №4:

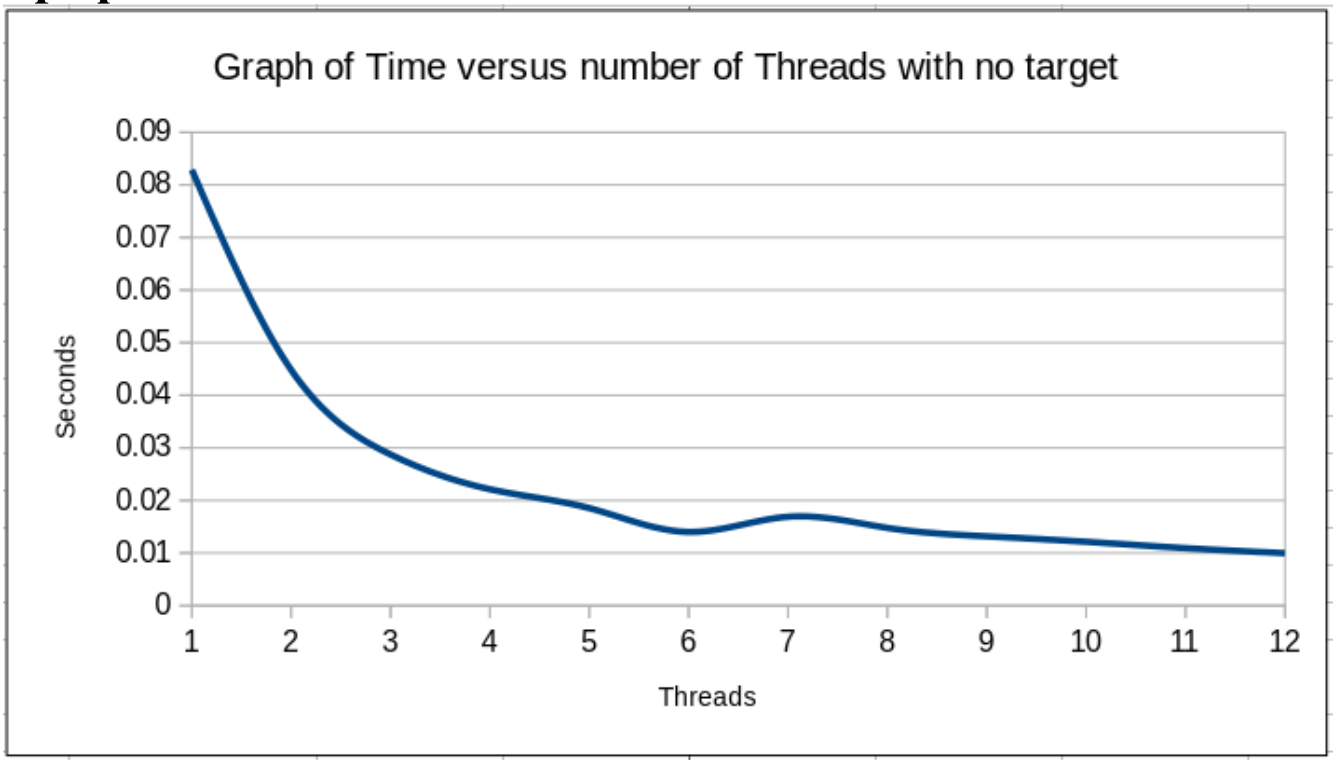


График №5:

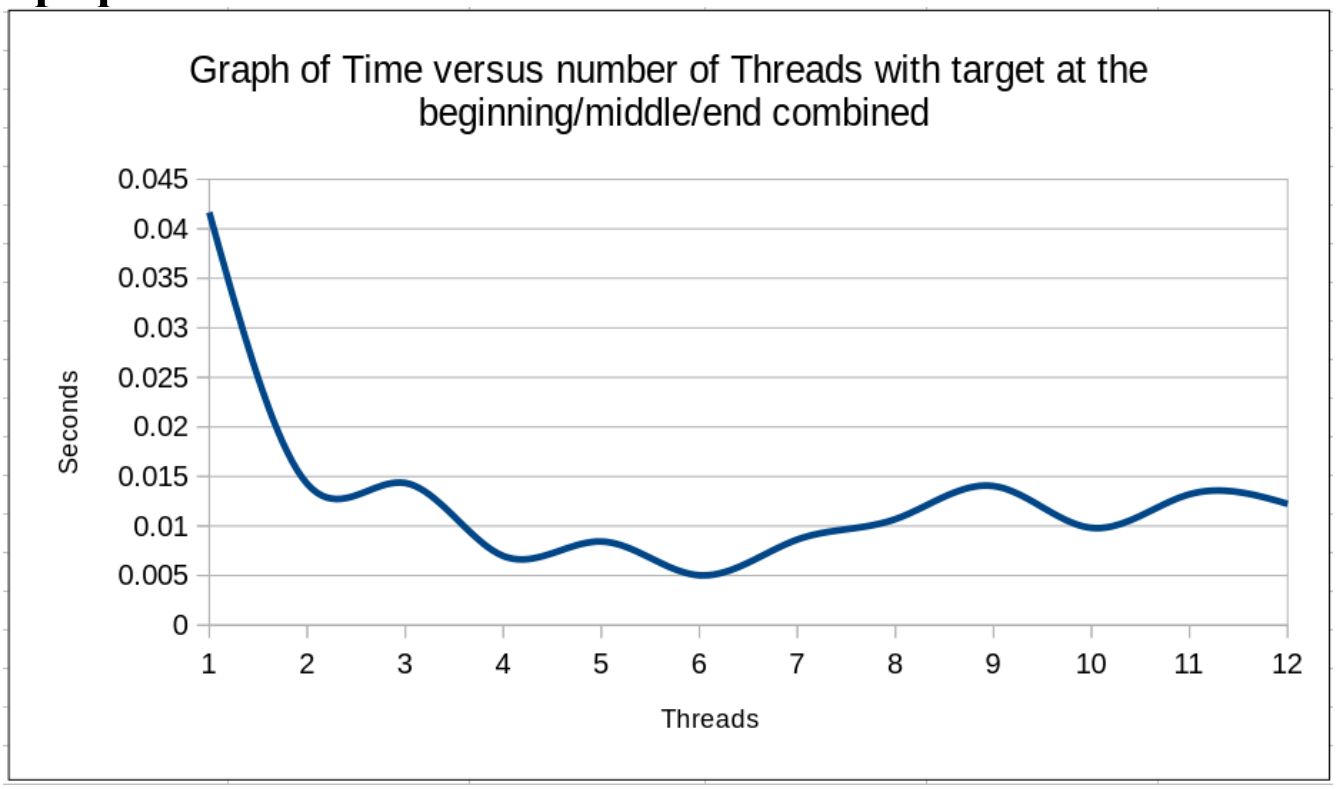


График №6:

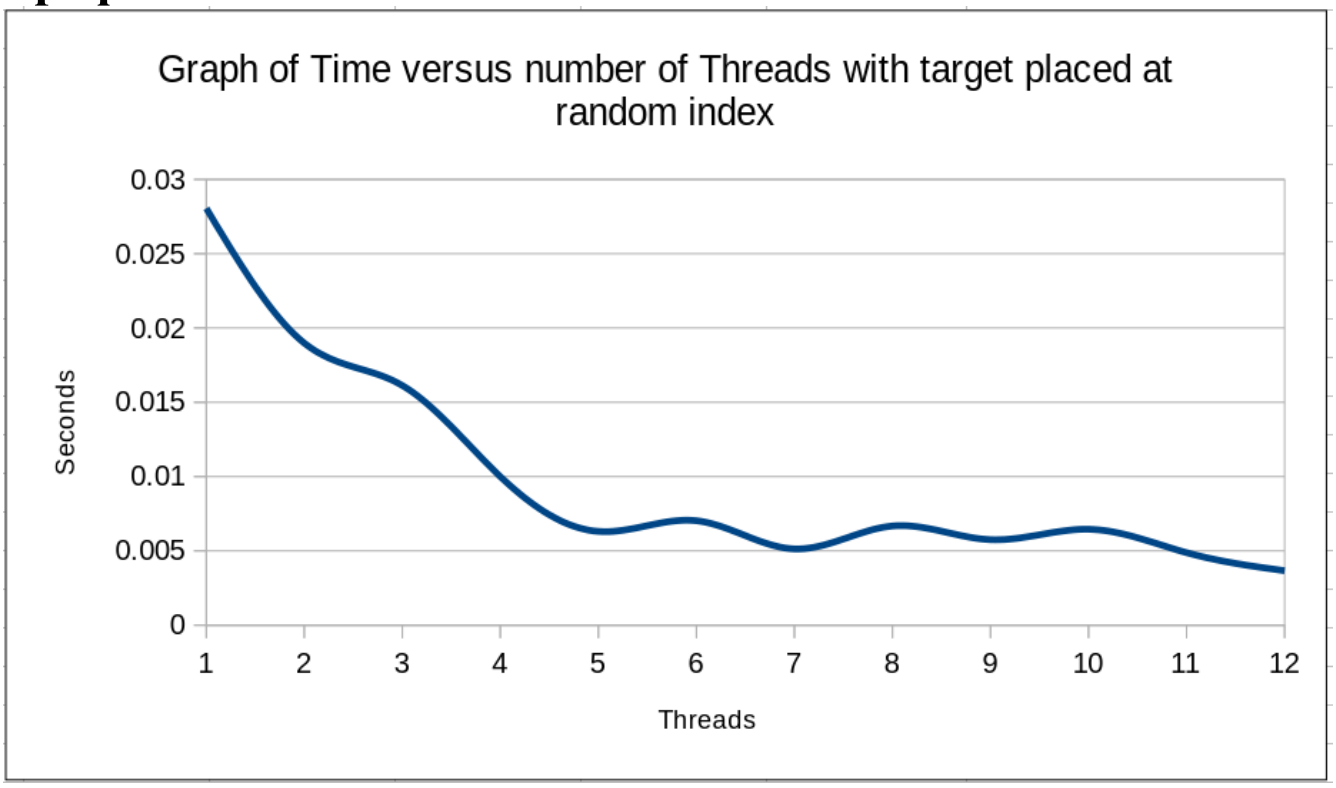


График №7:

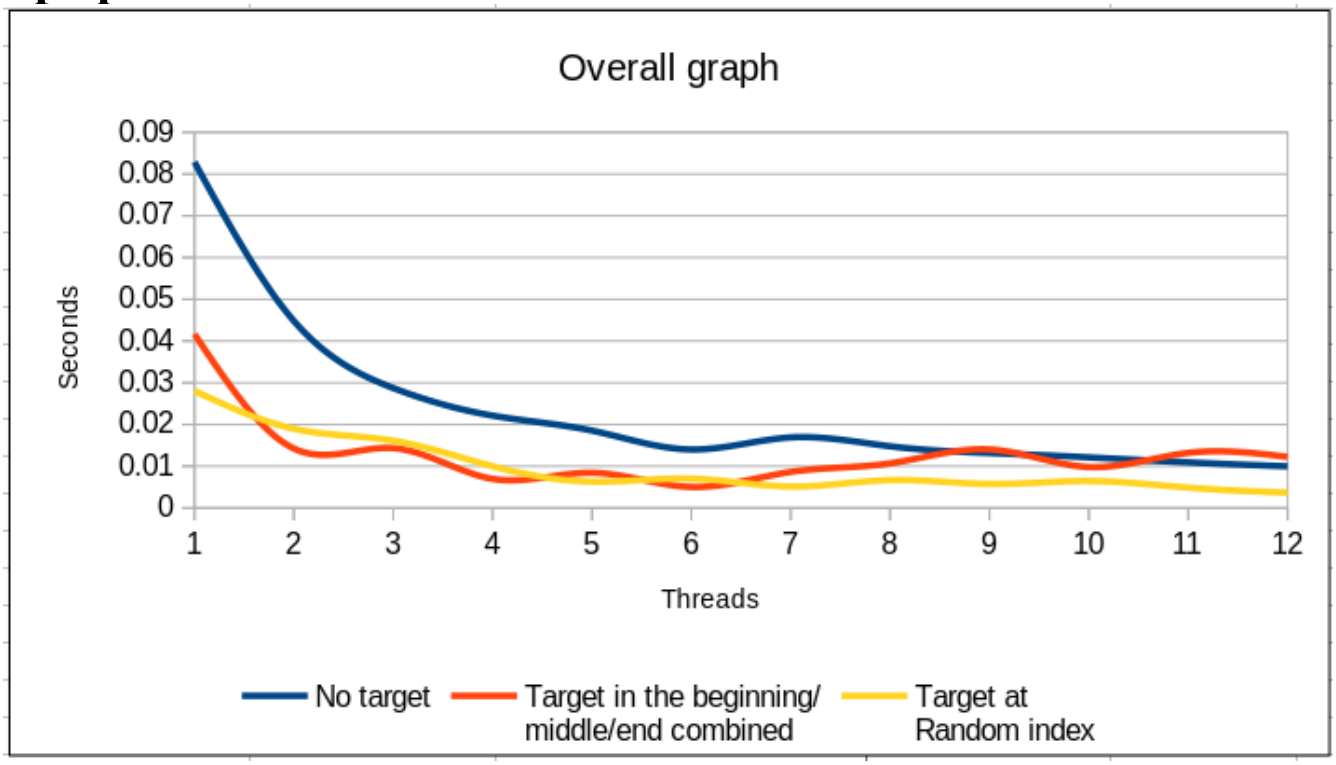


Таблица с данными:

Threads	Average Time From 10 attempts	Acceleration	Efficiency	Target in the Beginning	Target in the Middle	Target in the End	Average Time
1	0.081178	0.6946771293	0.694677129271	0.000014	0.044386	0.080589	0.041663
2	0.0410867	1.3725244422	0.686262221108	0.000236	0.000188	0.042302	0.014242
3	0.0275374	2.0478512859	0.682617095296	0.000129	0.014697	0.028201	0.01434233333333
4	0.0208629	2.7030038969	0.675750974217	0.000141	0.000236	0.020532	0.00696966666667
5	0.0174949	3.2233679529	0.644673590589	0.000125	0.008821	0.01637	0.00843866666667
6	0.0142182	3.9662193527	0.661036558777	0.000143	0.000218	0.01472	0.005027
7	0.0169047	3.3359065822	0.47655808317	0.000132	0.006466	0.019321	0.00863966666667
8	0.01491	3.782193159	0.472774144869	0.000108	0.000211	0.031765	0.01069466666667
9	0.0133588	4.2213746744	0.469041630486	0.000115	0.007419	0.034581	0.01403833333333
10	0.0128958	4.3729353743	0.437293537431	0.000118	0.000228	0.029089	0.00981166666667
11	0.0114249	4.9359294173	0.44872085612	0.000117	0.007474	0.032022	0.01320433333333
12	0.0111496	5.0578047643	0.421483730358	0.000122	0.007385	0.029214	0.01224033333333
Attempt #1	**Attempt #2**	**Attempt #3**	**Attempt #4**	**Target at Random index**	**No target**	**Target in the beginning/ middle/end combined**	**Target at Random index**
0.013848	0.034971	0.01832	0.045076	0.02805375	0.082885	0.041663	0.02805375
0.014128	0.035192	0.018477	0.008125	0.0189805	0.044818	0.014242	0.0189805
0.01401	0.011505	0.018811	0.020195	0.01613025	0.028735	0.0143423333333333	0.01613025
0.014136	0.017207	0.00045	0.008195	0.009997	0.022127	0.00696966666666667	0.009997
0.014129	0.006305	0.003941	0.000868	0.00631075	0.018535	0.00843866666666667	0.00631075
0.001964	0.011104	0.006403	0.008657	0.007032	0.014008	0.005027	0.007032
0.003556	0.004185	0.008115	0.004676	0.005133	0.016904	0.00863966666666667	0.005133
0.0052	0.00811	0.000444	0.012951	0.00667625	0.014777	0.0106946666666667	0.00667625
0.009378	0.00473	0.002384	0.006528	0.005755	0.013169	0.0140383333333333	0.005755
0.010543	0.009938	0.004013	0.001327	0.00645525	0.012136	0.00981166666666667	0.00645525
0.001193	0.002428	0.008378	0.007528	0.00488175	0.010929	0.0132043333333333	0.00488175
0.002875	0.007976	0.000531	0.003289	0.00366775	0.009979	0.0122403333333333	0.00366775

Код программы:

main.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>

int main()
{
    const int count = 10000000;
    const int random_seed = 92021447;
    const int target = 227;
    const int max_threads = 12;
    int index = -1;
    int found = 0;

    srand(time(NULL));

    int *array = (int *)malloc(count * sizeof(int));
    for (int i = 0; i < count; i++) { array[i] = rand(); }


    for (int threads = 1; threads <= max_threads; threads++)
    {
        found = 0;
        double start_time = omp_get_wtime();

        #pragma omp parallel num_threads(threads) shared(array, count, target, threads, found) reduction(max : index) default(none)
        {
            int length = count/threads;

            for (int i = length * omp_get_thread_num(); i < length * (omp_get_thread_num() + 1); i++) {
                if (!found && array[i] == target) {
                    index = i;
                    found = 1;
                    break;
                }
                if (found) {
                    break;
                }
            }
        }

        if (index == -1) {
            printf("Target not found. Threads: %d. Time: %f\n", threads, omp_get_wtime() - start_time);
        } else {
            printf("Found number %d at index %d. Threads: %d. Time: %f\n", target, index, threads, omp_get_wtime() - start_time);
        }
    }
    return (0);
}
```

Заключение

В ходе данного исследования была разработана параллельная программа для поиска заданного значения в массиве. Первоначально, на основе последовательной версии алгоритма, были построены параллельные версии, и было произведено время выполнения программы с разным числом запущенных потоков.

В результате работы программы была получена зависимость среднего времени выполнения T(n). Также была вычислена зависимость ускорения от числа потоков по формуле: A(n) = T(n)/ T(1). После этого была расчитана зависимость эффективности от числа потоков по формуле: E(n) = A(n)/n, Где T – время (в секундах) выполнения программы, n – количество потоков, A – ускорение, E – эффективность.

При добавлении дополнительных потоков уровень эффективности снижается, и этот спад постепенно увеличивается с увеличением числа потоков. Эффективность начинается с 0.6947 и уменьшается до 0.4215. После того, как число потоков достигает 7, эффективность резко снижается на 0.185, после чего изменяется в пределах [0; 0.05]. При добавлении дополнительных потоков ускорение увеличивается. Начальное значение ускорения составляет 0.6947, а максимальное достигает 5.0578. При 7 потоках также происходит спад и ускорение падает с 3.9662 до 3.3359. После этого с увеличением числа потоков ускорение монотонно растет до 5.0578.

Исходя из полученных данных, мною была рассчитаны ускорение и эффективность алгоритма для разного числа потоков и построены соответствующие графики зависимости времени выполнения, ускорения и эффективности от числа запущенных потоков. Результаты показали, что увеличение числа потоков способствует ускорению программы, но каждый дополнительный поток делает это менее эффективным.

Дополнительно, я рассмотрел влияние вероятности появления заданного элемента в массиве на время выполнения алгоритма. Путем сравнения разных сценариев, включая случаи, когда элемент отсутствует в массиве, и когда он гарантированно встречается, а также его положение в массиве, я установил, что увеличение вероятности появления элемента уменьшает время выполнения алгоритма.

Таким образом, результаты исследования позволяют сделать вывод, что эффективность параллельных алгоритмов зависит от количества потоков и особенностей данных, с которыми они работают.