Национальный исследовательский ядерный университет «МИФИ»

(Московский Инженерно–Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №2

«Выделение ресурса параллелизма. Технология OpenMP»

Тимин Александр Б21-515 (2023г.)

Рабочая среда:

- Процессор: AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz, 8 ядер (16 логических)
- Оперативная память: 16.0 GB DDR4 3200 МГц
- ОС: Windows 10 Pro 22H2 64-bit operating system, x64-based processor
- Среда разработки: Microsoft Visual Studio 2022 (v143)
- Версия OpenMP: 200203 (/openmp:llvm)

Последовательный алгоритм

Создается массив псевдослучайных чисел (array[count]). Задаем начальное (невозможное) значение для переменной index (index = count). В цикле обходим его, сравнивая каждый элемент с искомым (target): если сравниваемый элемент совпадает с искомым (array[i] == target), то выходим из цикла (break), предварительно сохранив индекс найденного элемента (index = i); иначе переходим к следующей итерации цикла. Если после выхода из цикла имеем index == count, то полагаем, что в массиве отсутствует целевой элемент.

Сложность алгоритма:

- в худшем случае (искомый элемент находится в конце массива или отсутствует – i >> 1) – O(count);
- в лучшем случае (искомый элемент находится в начале массива – array[i] == target и i << count) – O(1).

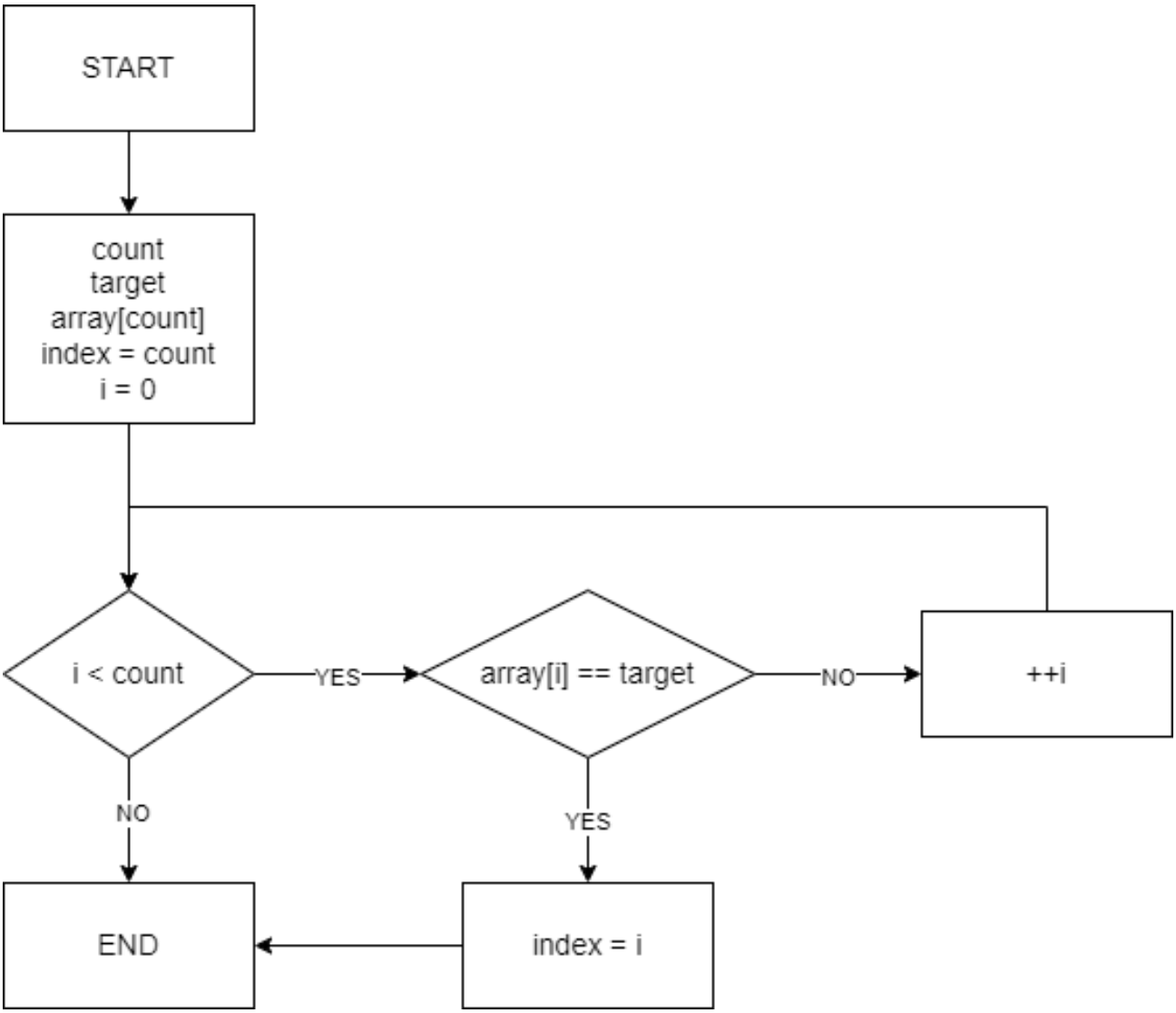
Параллельный алгоритм

Параллельный алгоритм отличается от последовательного только способом обработки цикла: все итерации автоматически распределяются между используемыми потоками, внутри которых обсчитываются отдельно; по завершении работы всех потоков происходит синхронизация, во время которой на основной поток передается минимальное из найденных значений переменной index. Весь остальной алгоритм идентичен последовательной версии.

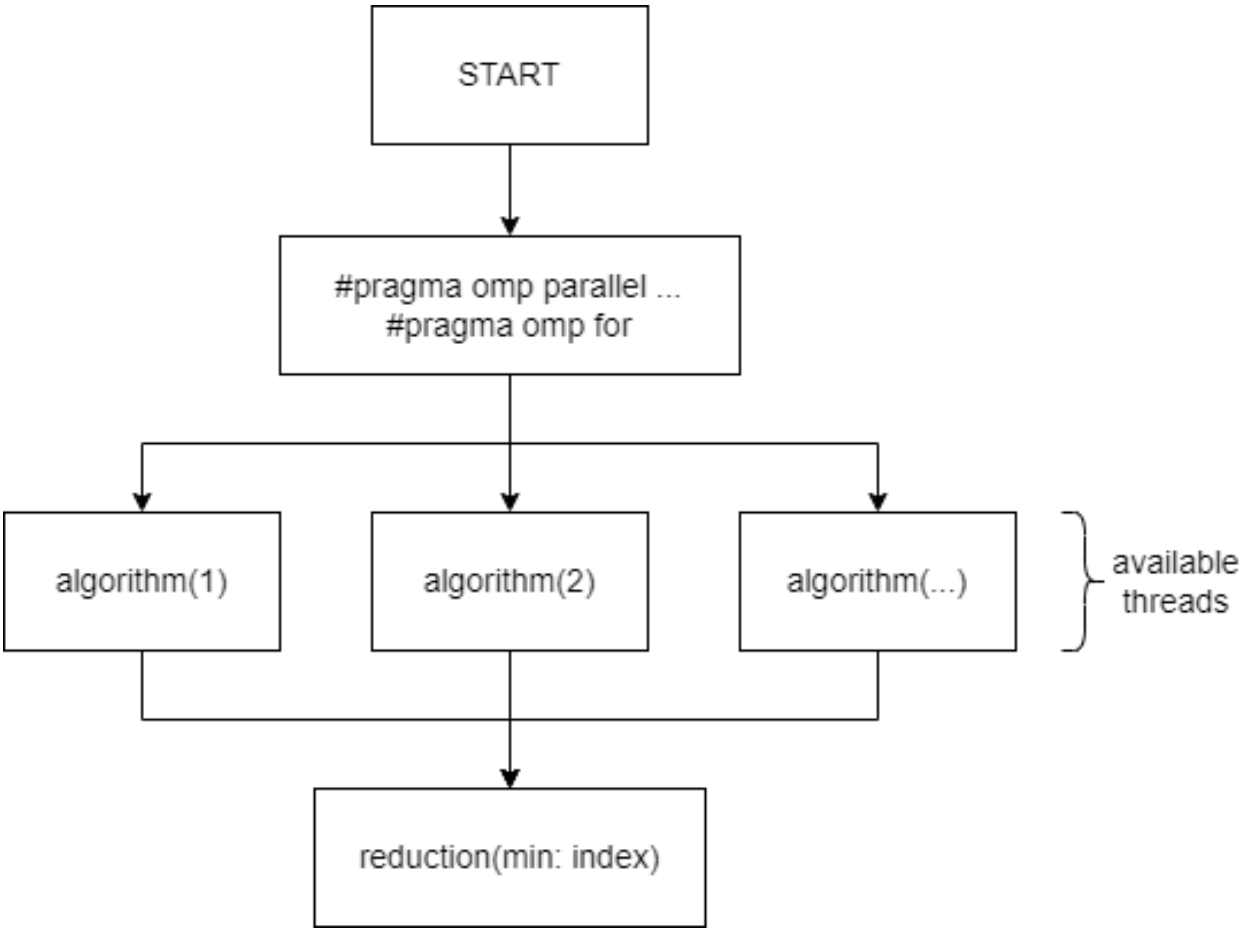
Сложность алгоритма:

- в худшем случае (искомый элемент будет найден в последних итерациях или не будет найден) – O(count/threads);
- в лучшем случае (искомый элемент будет найден в первых итерациях) – O(1/threads).

Блок-схема последовательного алгоритма



Блок-схема параллельного алгоритма



OpenMP

```
21
22      #pragma omp parallel num_threads(threads) reduction(min: index) private(i)
23      {
```

- **omp parallel** – показывает, что следующий блок кода будет выполняться с использованием нескольких потоков;
- **num_threads(threads)** – задает количество потоков, равное **threads**;
- **reduction(min: index)** – определяет значение переменной **index** на главном потоке после завершения параллельного участка как минимальное из всех значений;
- **private(i)** – показывает, что переменная **i** является частной для каждого потока (после завершения параллельного участка считается неопределенной). Если данный параметр не будет указан, то переменная **i** будет иметь правила по умолчанию (в данном случае она будет считаться общей, так как была объявлена вне параллельного блока).

```
23      {
24          #pragma omp for
```

- **omp for** – показывает, что следующий цикл **for** будет выполняться параллельно несколькими потоками (необходимо: #include <omp.h>).

```
35          end = omp_get_wtime();
```

- **omp_get_wtime()** – возвращает астрономическое время прошедшее с определенного момента в прошлом (необходимо: #include <omp.h>.

```
55          available_threads = omp_get_num_procs();
```

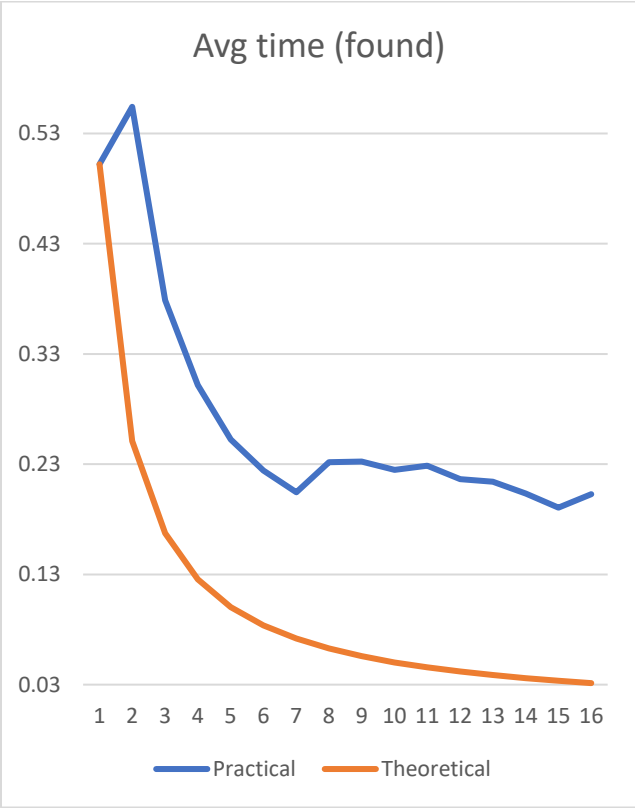
- **omp_get_num_procs() –** возвращает количество доступных процессоров (необходимо: #include <omp.h>).

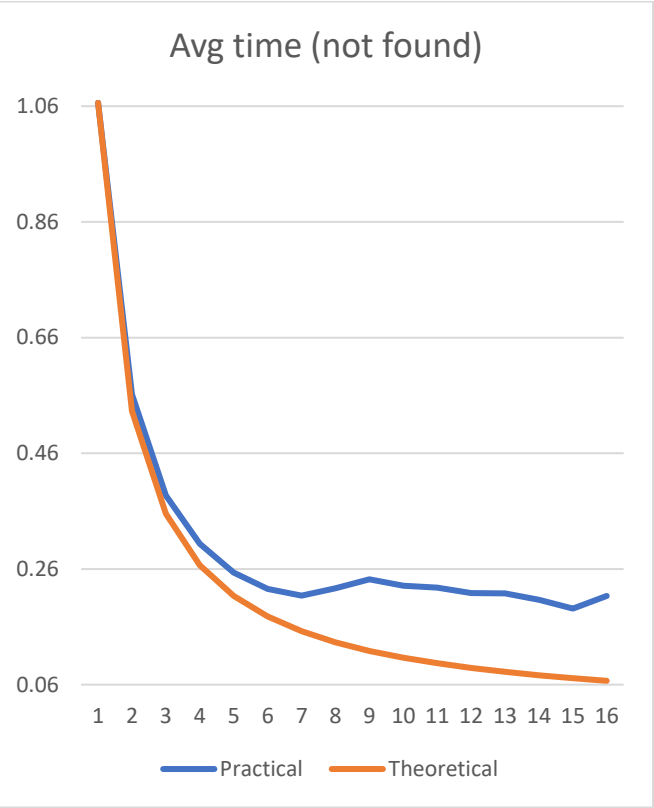
Таблицы данных

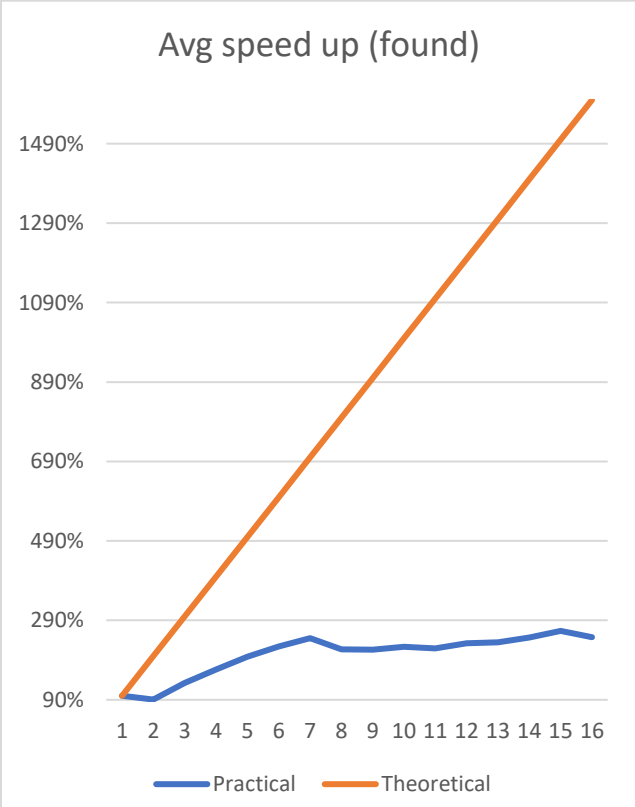
Target was found						
Threads	Avg time (pr)	Avg time (th)	Avg speed up (pr)	Avg speed up (th)	Avg efficiency (pr)	Avg efficiency (th)
1	0.502100	0.502100	100.00%	100.00%	100.00%	100.00%
2	0.554204	0.251050	90.60%	200.00%	45.30%	100.00%
3	0.378761	0.167367	132.56%	300.00%	44.19%	100.00%
4	0.301739	0.125525	166.40%	400.00%	41.60%	100.00%
5	0.252510	0.100420	198.84%	500.00%	39.77%	100.00%
6	0.224277	0.083683	223.88%	600.00%	37.31%	100.00%
7	0.204631	0.071729	245.37%	700.00%	35.05%	100.00%
8	0.231664	0.062763	216.74%	800.00%	27.09%	100.00%
9	0.232441	0.055789	216.01%	900.00%	24.00%	100.00%
10	0.224735	0.050210	223.42%	1000.00%	22.34%	100.00%
11	0.228712	0.045645	219.53%	1100.00%	19.96%	100.00%
12	0.216338	0.041842	232.09%	1200.00%	19.34%	100.00%
13	0.214151	0.038623	234.46%	1300.00%	18.04%	100.00%
14	0.203393	0.035864	246.86%	1400.00%	17.63%	100.00%
15	0.190575	0.033473	263.47%	1500.00%	17.56%	100.00%
16	0.202719	0.031381	247.68%	1600.00%	15.48%	100.00%

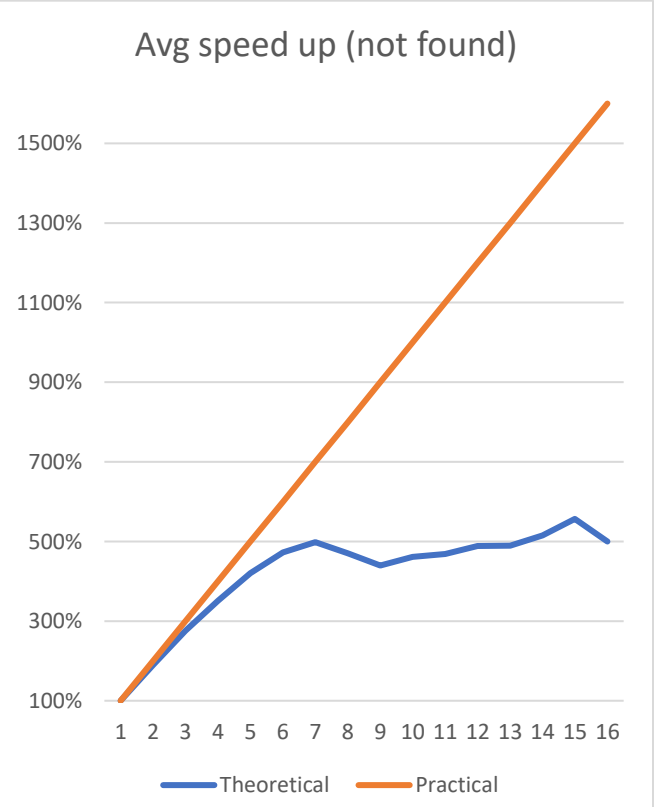
Target was not found						
Threads	Avg time (pr)	Avg time (th)	Avg speed up (pr)	Avg speed up (th)	Avg efficiency (pr)	Avg efficiency (th)
1	1.065946	1.065946	100.00%	100.00%	100.00%	100.00%
2	0.561709	0.532973	189.77%	200.00%	94.88%	100.00%
3	0.386784	0.355315	275.59%	300.00%	91.86%	100.00%
4	0.303679	0.266486	351.01%	400.00%	87.75%	100.00%
5	0.253444	0.213189	420.58%	500.00%	84.12%	100.00%
6	0.225357	0.177658	473.00%	600.00%	78.83%	100.00%
7	0.214018	0.152278	498.06%	700.00%	71.15%	100.00%
8	0.226740	0.133243	470.12%	800.00%	58.76%	100.00%
9	0.242280	0.118438	439.96%	900.00%	48.88%	100.00%
10	0.231012	0.106595	461.42%	1000.00%	46.14%	100.00%
11	0.227491	0.096904	468.57%	1100.00%	42.60%	100.00%
12	0.218213	0.088829	488.49%	1200.00%	40.71%	100.00%
13	0.217723	0.081996	489.59%	1300.00%	37.66%	100.00%
14	0.206918	0.076139	515.15%	1400.00%	36.80%	100.00%
15	0.191466	0.071063	556.73%	1500.00%	37.12%	100.00%
16	0.213187	0.066622	500.00%	1600.00%	31.25%	100.00%

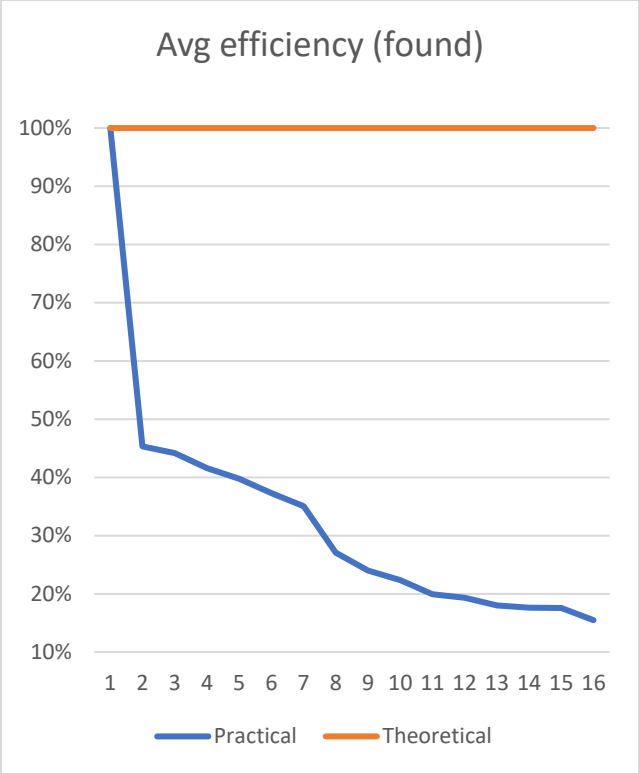
Графики

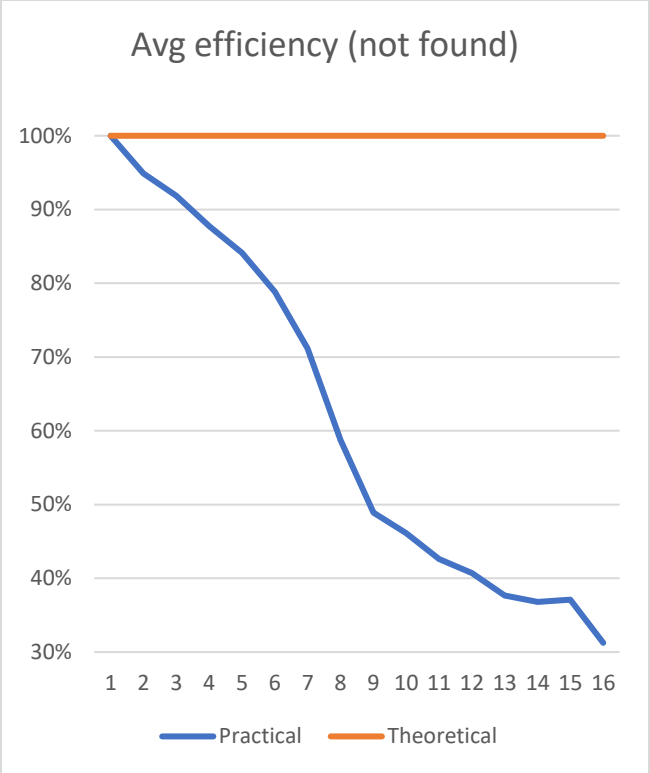

Avg time (found)
0.53
0.43
0.33
0.23
0.13
0.03
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical


Avg time (not found)
1.06
0.86
0.66
0.46
0.26
0.06
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical


Avg speed up (found)
1490%
1290%
1090%
890%
690%
490%
290%
90%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical


Avg speed up (not found)
1500%
1300%
1100%
900%
700%
500%
300%
100%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Theoretical
Practical


Avg efficiency (found)
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical


Avg efficiency (not found)
100%
90%
80%
70%
60%
50%
40%
30%
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Practical
Theoretical

Заключение

Был проанализирован алгоритм поиска элемента в целочисленном массиве (блок-схема). Была разработана и реализована при помощи директив OpenMP параллельная версия алгоритма (блок-схема).

Для массива псевдослучайных чисел (array[]) и массива псевдослучайных чисел-целей (targets[]) были замерены и вычислены временные характеристики работы параллельного алгоритма на различном количестве потоков: среднее время работы, среднее ускорение и средняя эффективность алгоритма.

Были построены графики соотношения теоретических значений характеристик в сравнении с практическими.

Приложение

- https://github.com/KATEHOK/par_prog-2/blob/main/diagram/algorithm.png – блок-схема последовательного алгоритма;
- https://github.com/KATEHOK/par_prog-2/blob/main/diagram/parallel.png – блок-схема параллельного алгоритма;
- https://github.com/KATEHOK/par_prog-2/blob/main/src/lab2.c – исходный код;
- https://github.com/KATEHOK/par_prog-2/blob/main/report/data.txt – «сырые» данные (текст);
- https://github.com/KATEHOK/par_prog-2/blob/main/report/lab2.xlsx – «обработанные» данные (таблицы и графики);
- https://github.com/KATEHOK/par_prog-2/blob/main/report/lab2.docx – отчет (docx);
- https://github.com/KATEHOK/par_prog-2/blob/main/report/lab2.pdf – отчет (pdf).