

# ОТЧЁТ

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

Группа  
Студент  
Преподаватель

Б21-525  
М.Р. Тимурович  
М.А. Куприяшин

# Оглавление

1.	Описание рабочей среды . . . . .	3
2.	Анализ приведенного алгоритма . . . . .	3
3.	Анализ временных характеристик последовательного алгоритма . . .	5
4.	Анализ временных характеристик параллельного алгоритма . . . . .	6
5.	Заключение . . . . .	14
6.	Приложение . . . . .	15

# 1. Описание рабочей среды

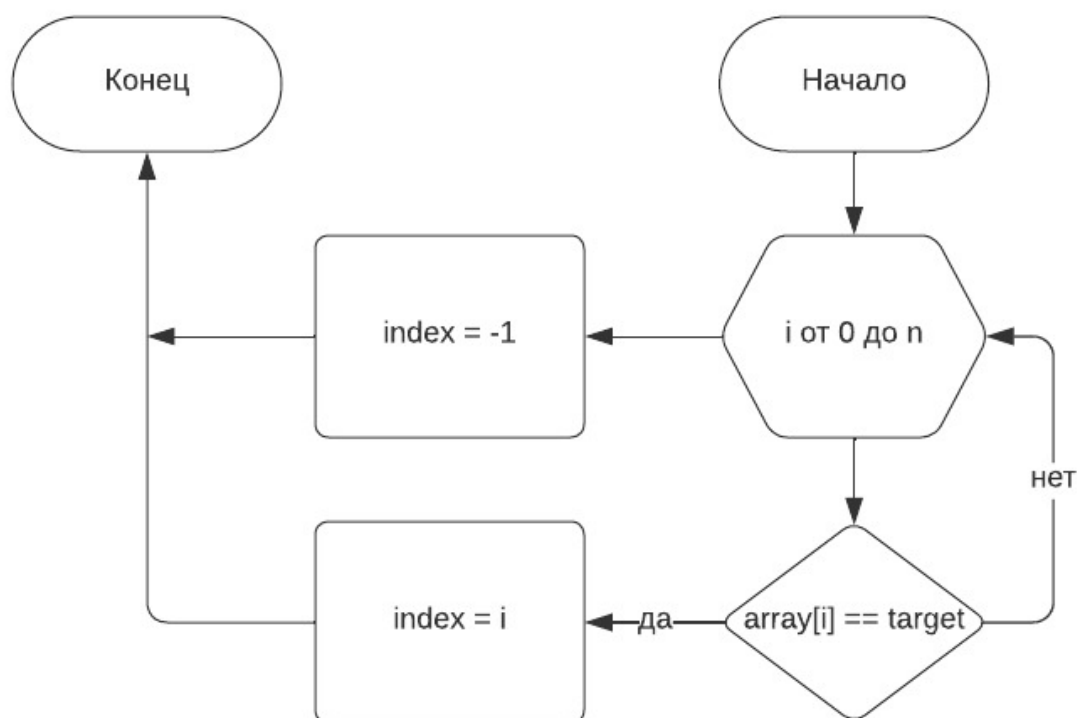
- Модель процессора:	Intel Core i3-10110U CPU @ 2.10GHz
- Число ядер:	4
- Архитектура:	x86-64
- ОС:	Linux, дистрибутив Ubuntu v20.04
- RAM объем:	2x8192 MB
- RAM тип:	DDR4
- Среда разработки:	Visual Studio Code
- Компилятор:	gcc v9.4.0
- Версия OpenMP:	201511

## 2. Анализ приведенного алгоритма

В задании лабораторной работы приведена программа, осуществляющая поиск заданного элемента в массиве.

Основное отличие от задания лабораторной работы 1 заключается в том, что при первом нахождении заданного элемента продолжать поиск не требуется.

### Блоксхема алгоритма



## Описание используемых директив OpenMP

**parallel** - определяет параллельную область, которая представляет собой код, который будет выполняться несколькими потоками параллельно. Директива **parallel** была объявлена со следующими атрибутами:

- **num\_threads()** - задаёт количество потоков в параллельном блоке (по умолчанию **parallel** использует все потоки);
- **shared()** - объявляет, что переменные должны быть общими между всеми потоками;

**for** - разделяет работу цикла между потоками (без неё каждый поток обрабатывал бы весь массив).

Действие директивы **parallel** распространяется на следующий блок программного кода:

```
#pragma omp for
for(int i = 0; i < count; i++) {
    if (flag) continue;

    if(array[i] == target) {
        #pragma omp critical
        {
            index = i;
            flag = true;
        }
    }
}
```

В свою очередь директива **for** действует на следующую строку:

```
for(int i = 0; i < count; i++)
```

**critical** - определяет раздел кода, который должен выполняться одним потоком за раз.

## Описание работы алгоритма

Директивой `parallel` объявляется блок кода, который будет исполняться параллельно. Далее внутри `for` потоки распределяют между собой итерации цикла. Каждый поток ищет элемент равный заданному. При нахождении такого элемента внутри `critical` переменной `index` присваивается найденное значение индекса. После этого потоки перестают искать элемент, пропуская итерации цикла с помощью `continue`.

## 3. Анализ временных характеристик последовательного алгоритма

### Описание эксперимента

- Эксперименты проводились на шести типах массивов
  - mode 0:** Заданный элемент находится на первой позиции
  - mode 1:** Заданный элемент находится в центре массива
  - mode 2:** Заданный элемент находится на последней позиции
  - mode 3:** Рандомный массив
  - mode 4:** Все элементы подходят
  - mode 5:** Ни один элемент не подходит
- Измеряется время работы алгоритма на 1 000 различных массивах длиной 10 000 000 элементов. Находится среднее значение;

### Экспериментальные показатели

- Среднее время работы последовательного алгоритма
  - mode 0:**  $O(1)$   $1.7 * 10^{-7}$  [с]
  - mode 1:**  $O(n)$  0.01300013 [с]
  - mode 2:**  $O(n)$  0.02465431 [с]
  - mode 3:**  $O(n)$  0.01212091 [с]
  - mode 4:**  $O(1)$   $2.3 * 10^{-7}$  [с]
  - mode 5:**  $O(n)$  0.0260270390 [с]

## 4. Анализ временных характеристик параллельного алгоритма

### Описание эксперимента

- Эксперименты проводились на шести типах массивов
  - mode 0:** Заданный элемент находится на первой позиции
  - mode 1:** Заданный элемент находится в центре массива
  - mode 2:** Заданный элемент находится на последней позиции
  - mode 3:** Рандомный массив
  - mode 4:** Все элементы подходят
  - mode 5:** Ни один элемент не подходит
- измеряется время работы алгоритма для одного и того же массива, но на разном числе потоков: от 1 до 10;
- измерения производятся для 1 000 различных массивов, размер массива 10 000 000 элементов.

## Результаты измерений

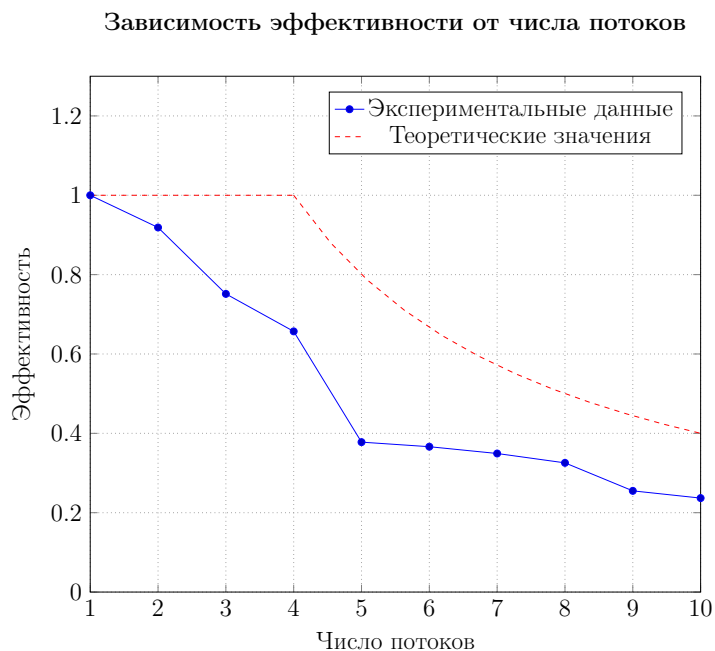
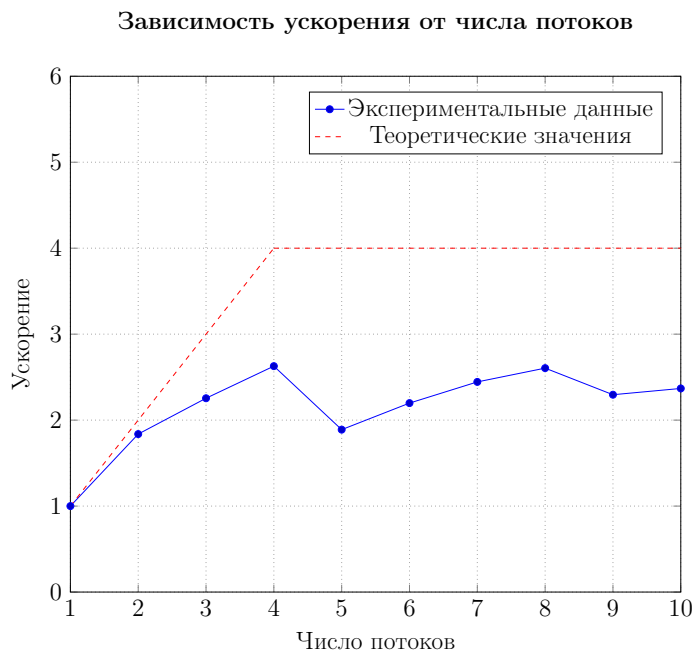
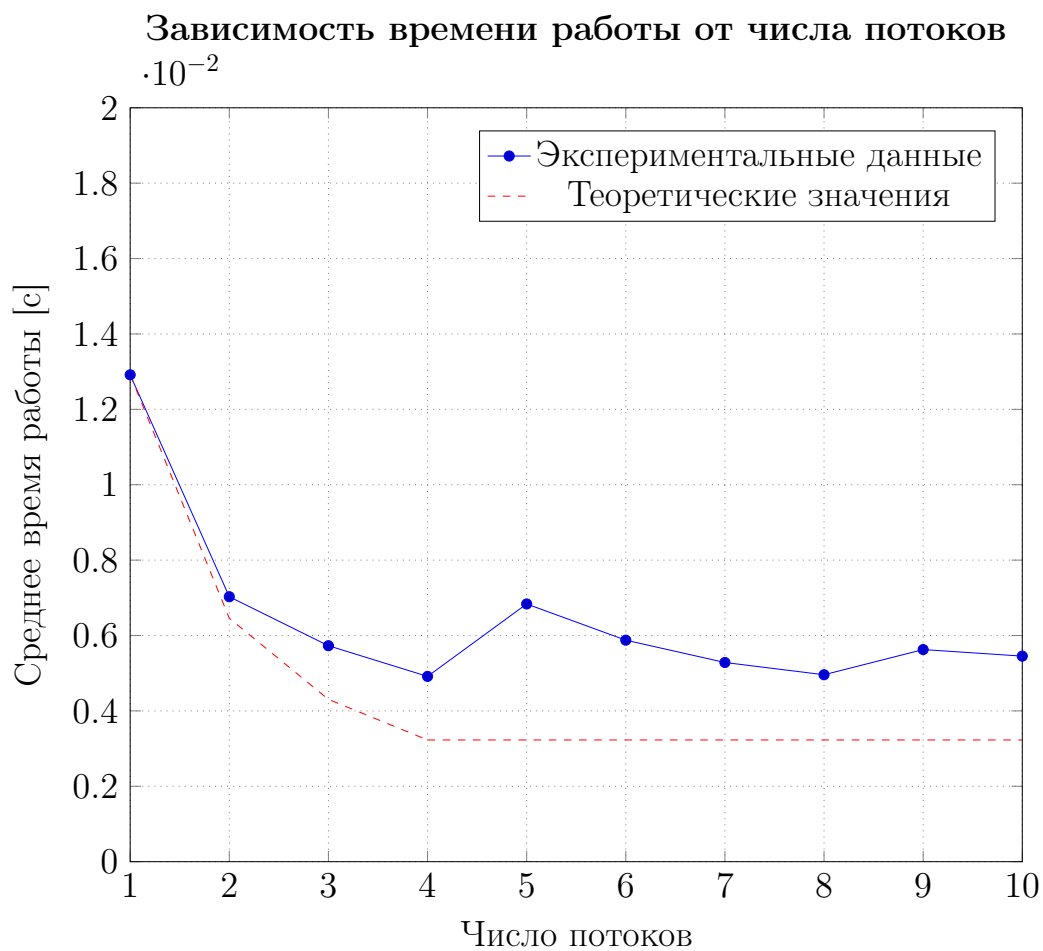
Следующие таблицы содержат полученные в результате эксперимента данные: среднее время работы для различного числа потоков и конфигураций массивов.

threads	mode 0	threads	mode 1	threads	mode 2
1	0.012916	1	0.017338	1	0.025239
2	0.007029	2	0.007039	2	0.013164
3	0.005729	3	0.007438	3	0.009254
4	0.004915	4	0.004929	4	0.007990
5	0.006837	5	0.008002	5	0.008930
6	0.005876	6	0.010619	6	0.008778
7	0.005284	7	0.007856	7	0.008453
8	0.004959	8	0.006534	8	0.007874
9	0.005627	9	0.007855	9	0.008262
10	0.005453	10	0.006564	10	0.008053

threads	mode 3	threads	mode 4	threads	mode 5
1	0.017379	1	0.012749	1	0.022069
2	0.009604	2	0.007006	2	0.012364
3	0.007436	3	0.005721	3	0.009657
4	0.006459	4	0.004946	4	0.007919
5	0.007828	5	0.006310	5	0.010624
6	0.007422	6	0.005899	6	0.009390
7	0.006848	7	0.005282	7	0.008564
8	0.006517	8	0.004964	8	0.008148
9	0.007118	9	0.005594	9	0.009218
10	0.006963	10	0.005449	10	0.008826

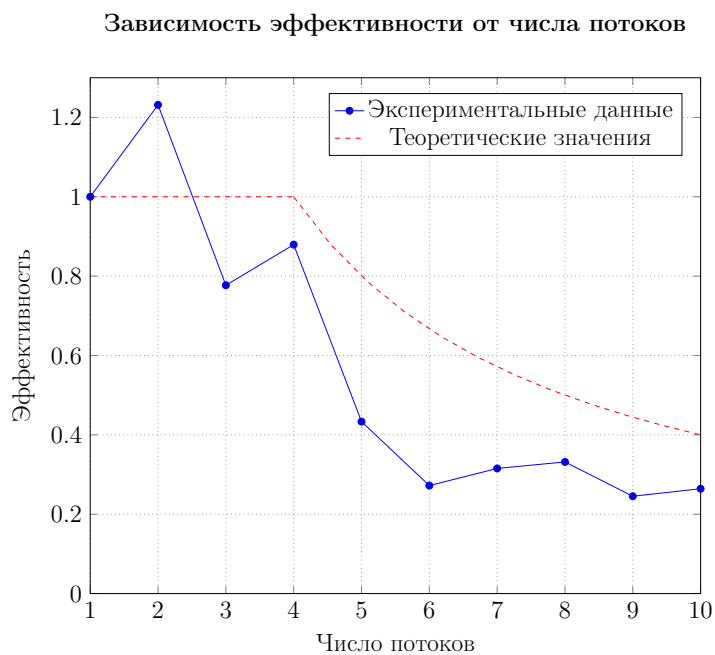
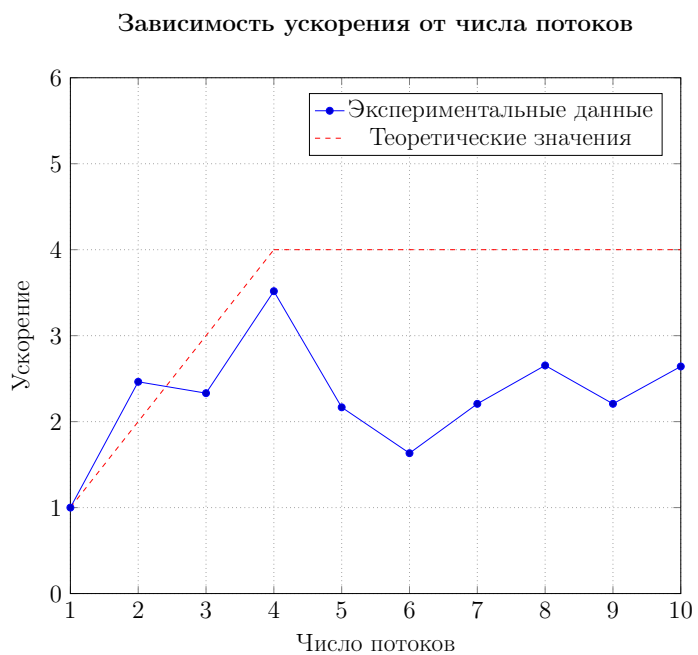
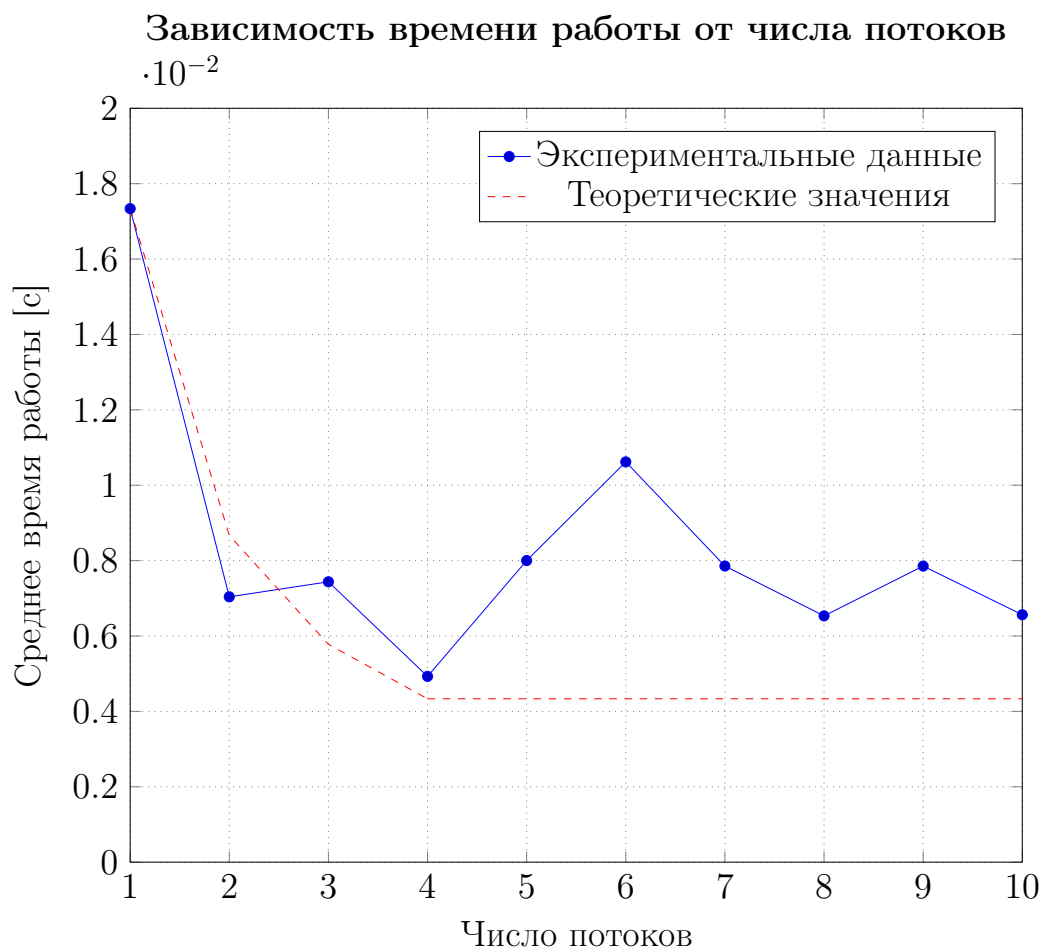
# Графики

## 1. Заданный элемент на первой позиции (mode 0)

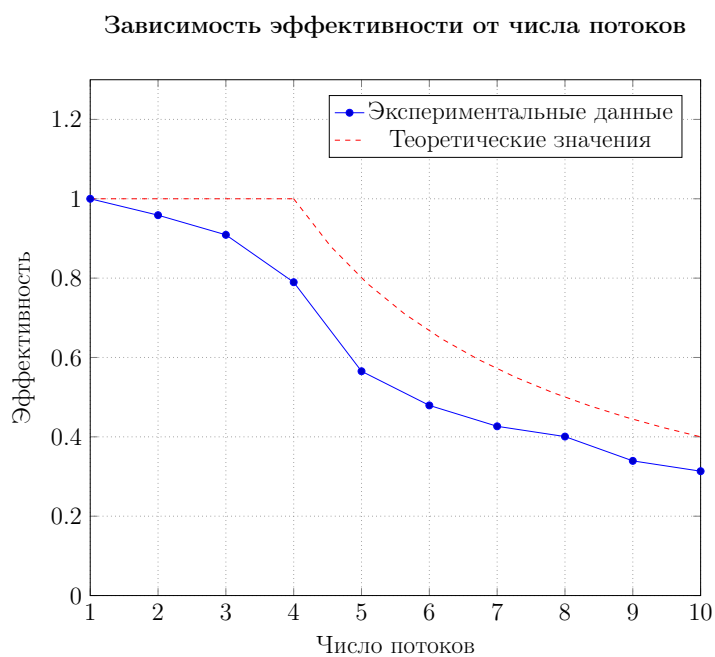
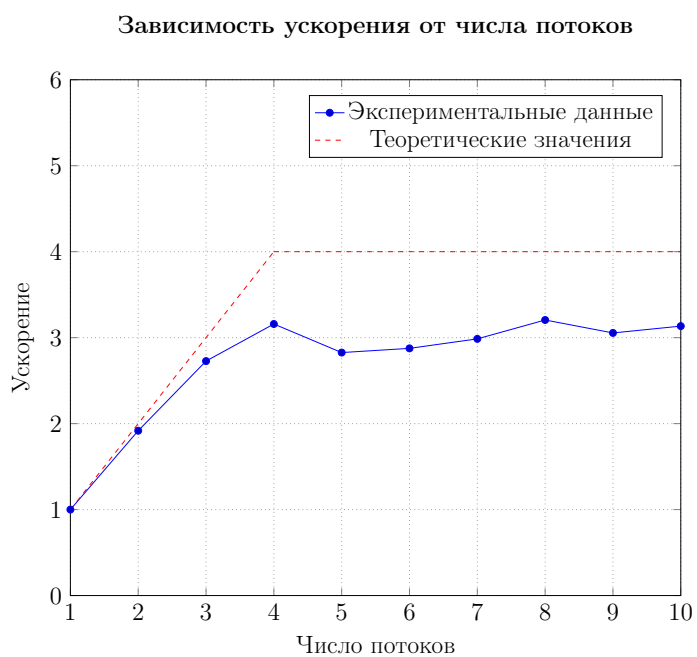
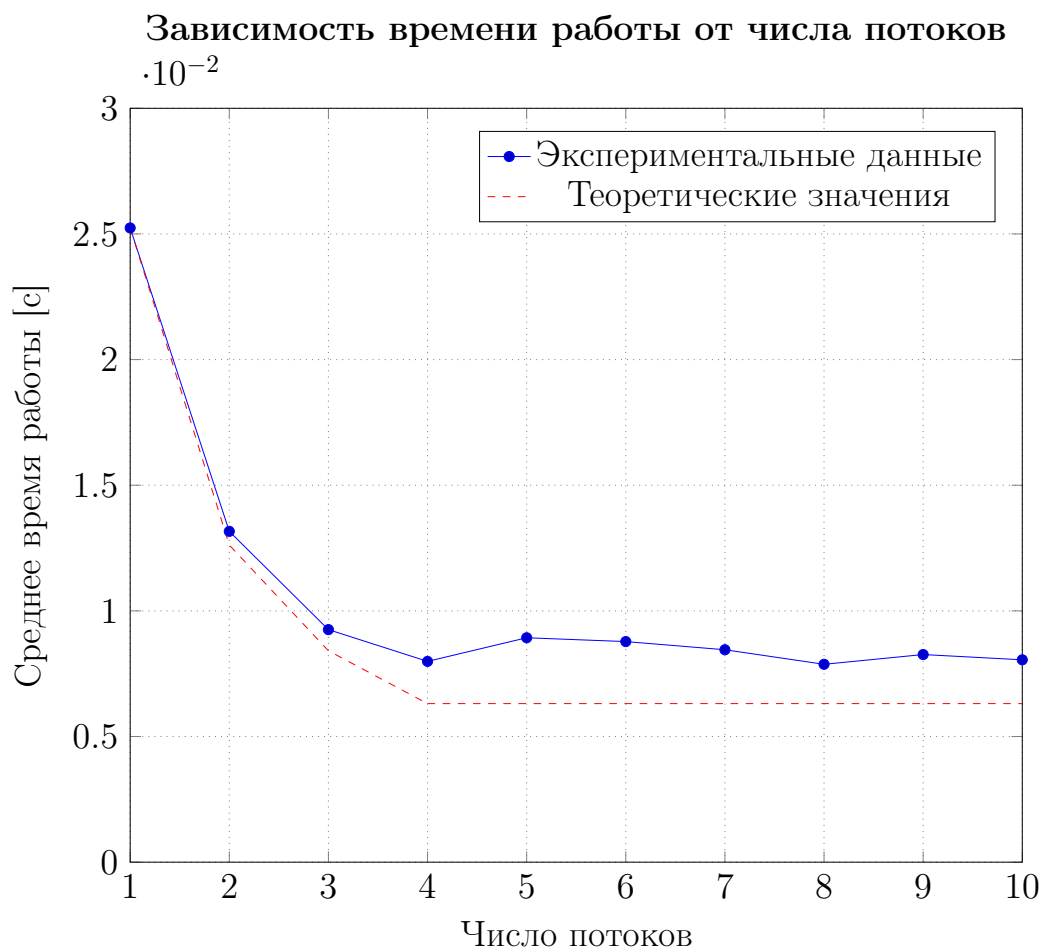




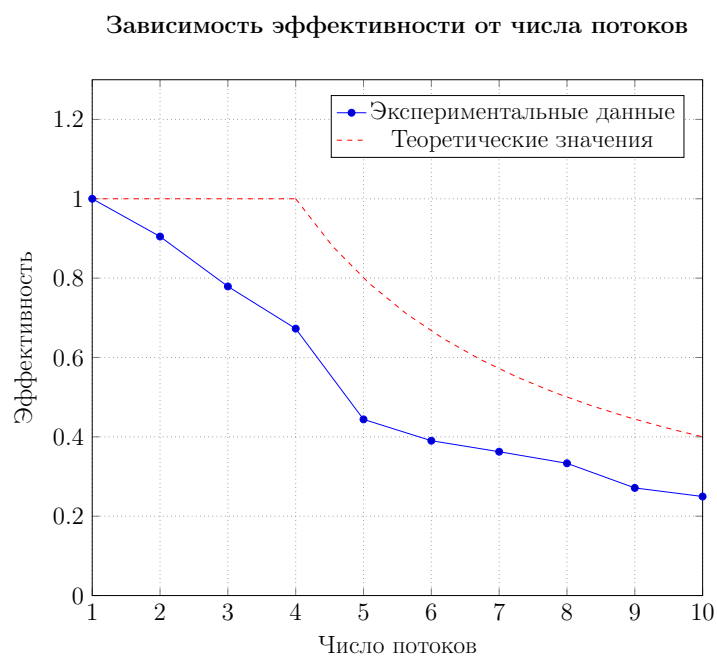
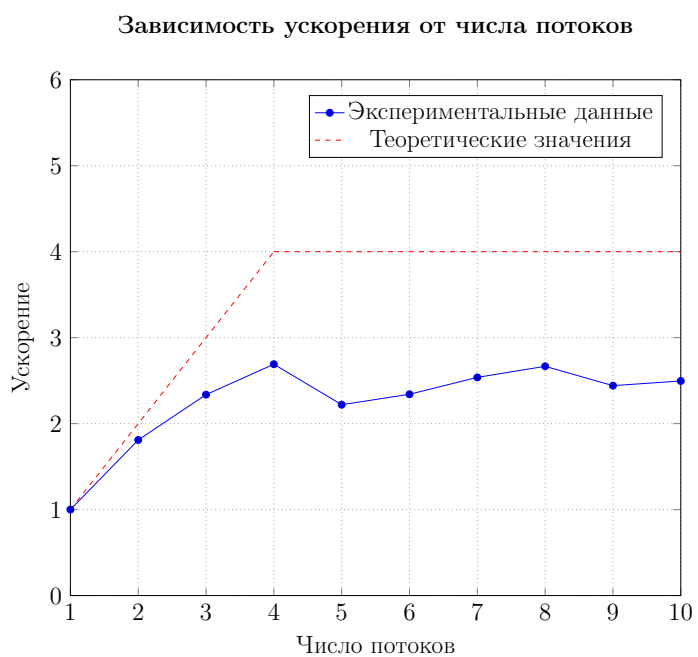
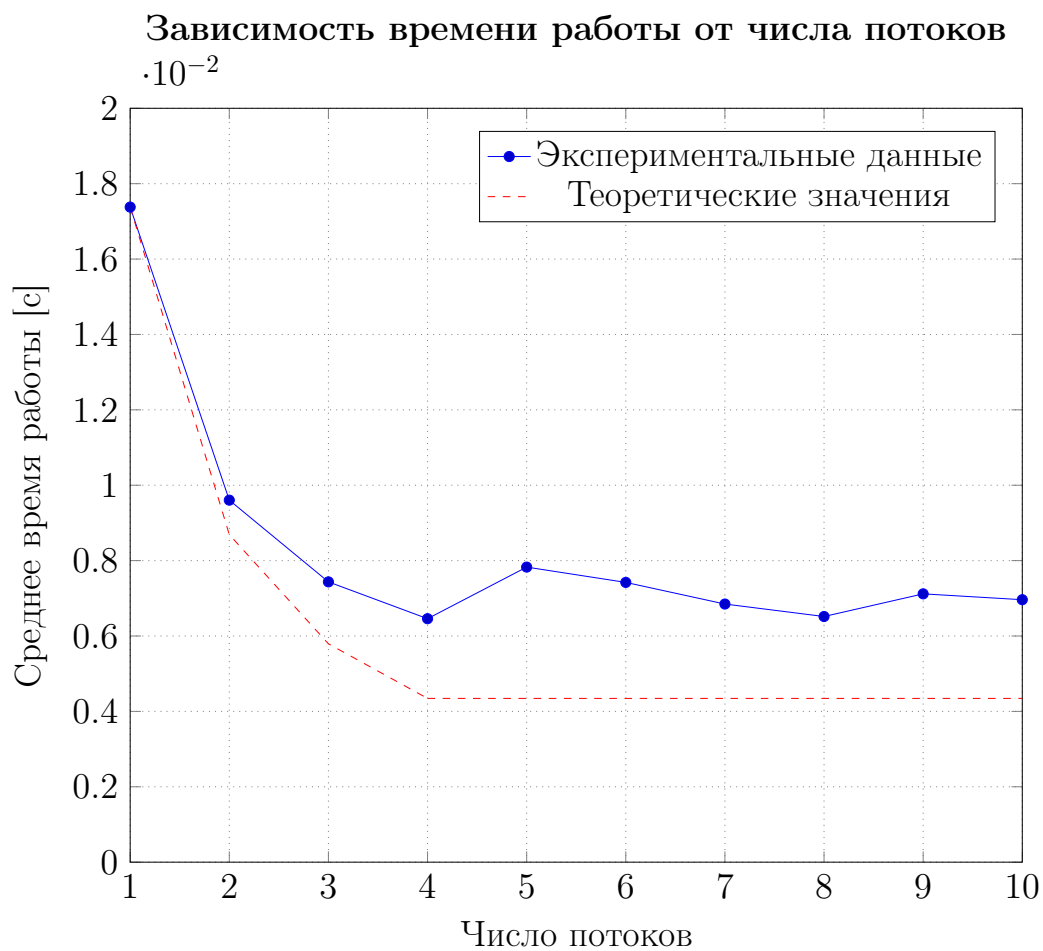
## 2. Заданный элемент в центре (mode 1)



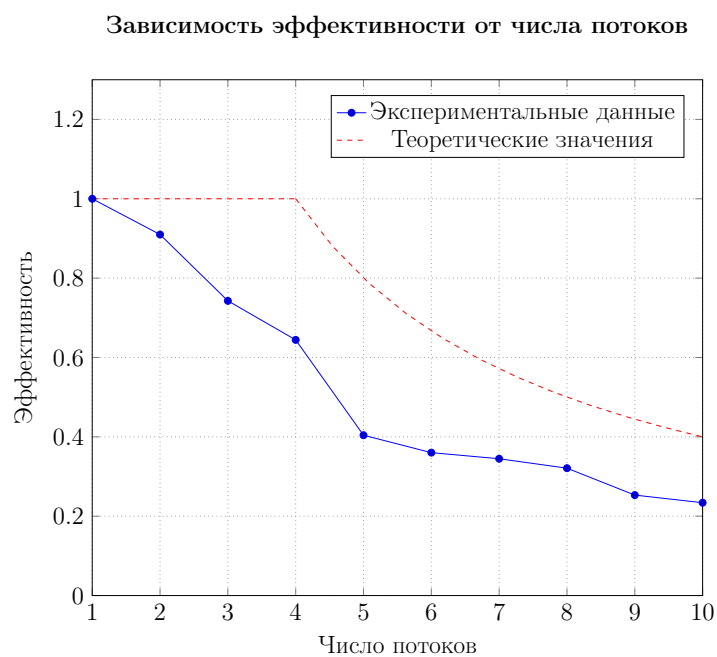
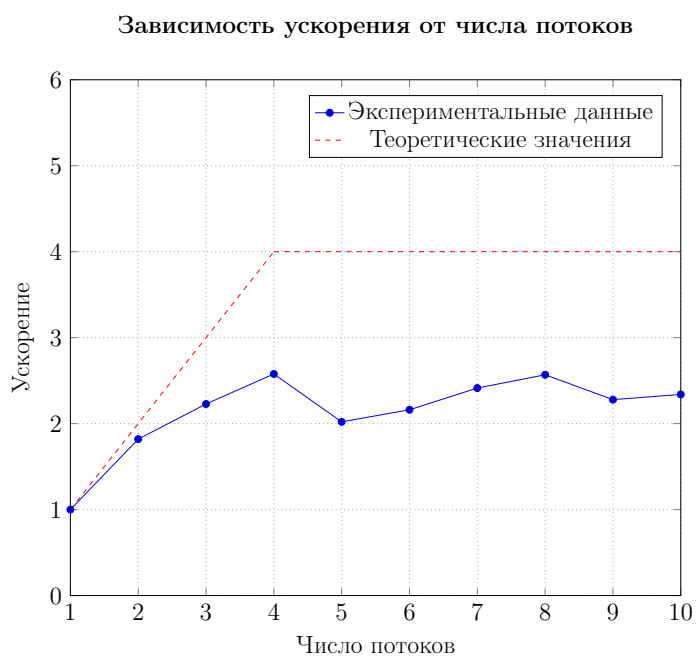
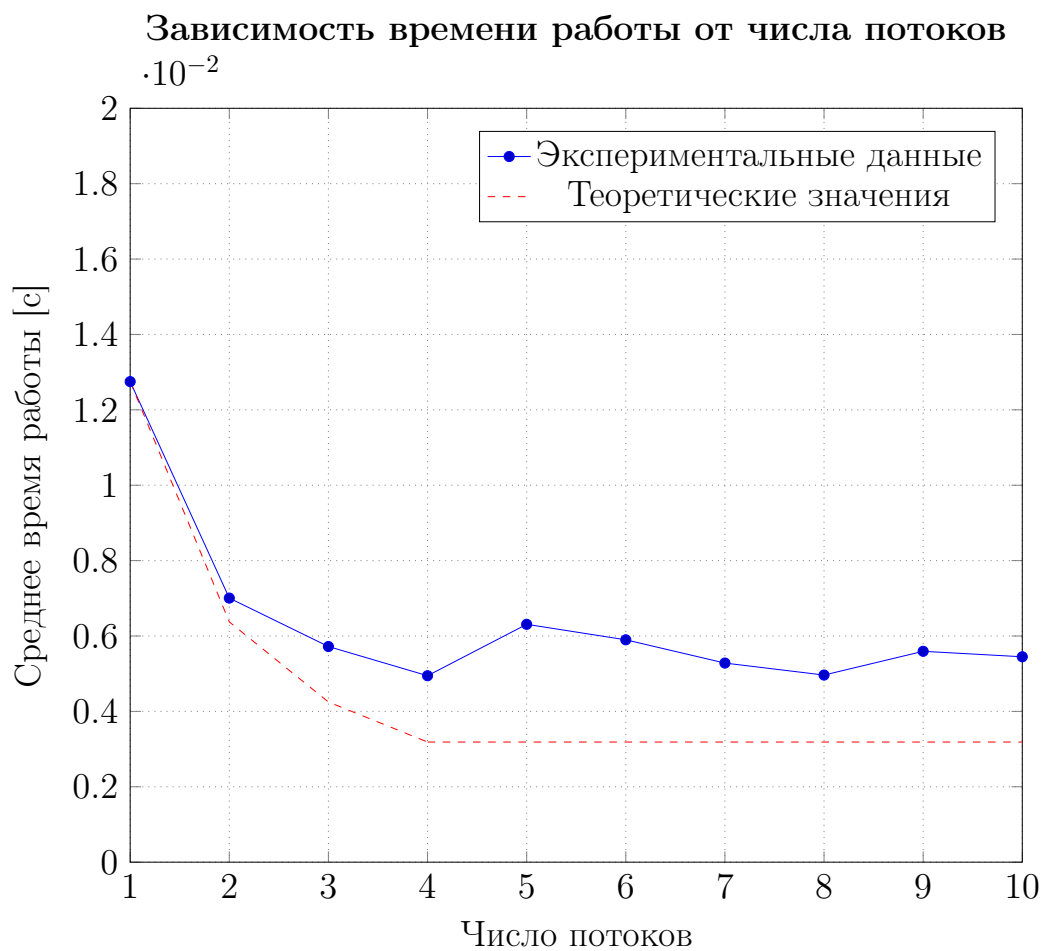
### 3. Заданный элемент на последней позиции (mode 2)



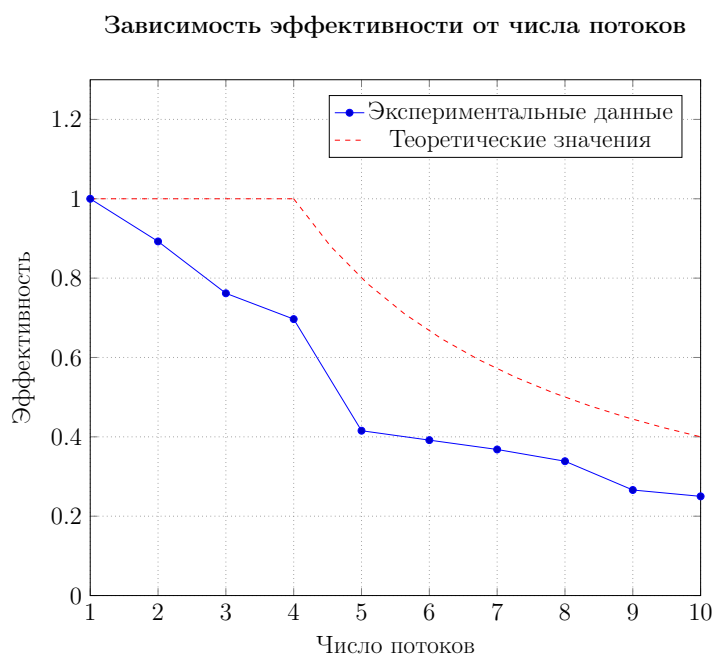
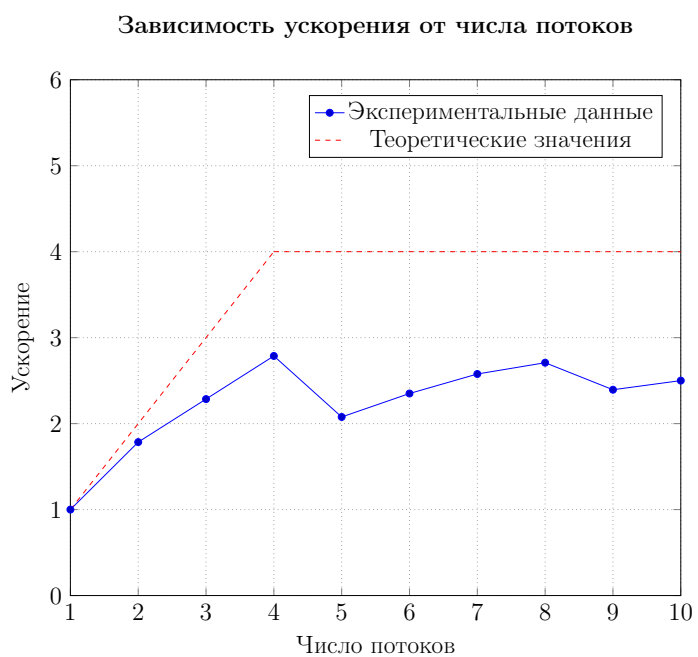
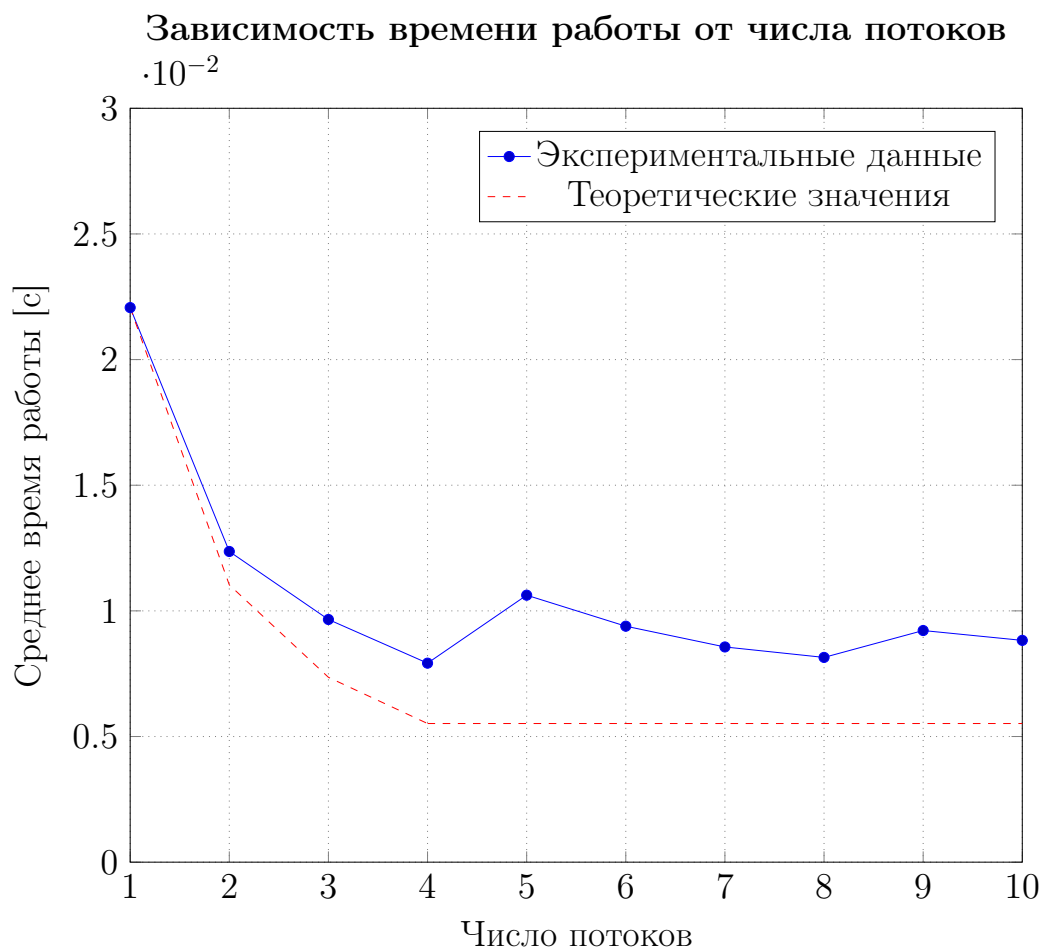
## 4. Рандомный массив (mode 3)



## 5. Все элементы подходят (mode 4)



## 6. Ни один элемент не подходит (mode 5)



## 5. Заключение

В ходе лабораторной работы было измерено время работы алгоритма поиска заданного элемента в массиве для различного числа потоков. По полученным данным были вычислены значения ускорения и эффективности для шести видов массивов. Построены соответствующие графики. Анализируя результаты эксперимента, можно обратить внимание на следующее:

- среднее время работы алгоритма уменьшается с ростом числа потоков до 4 включительно. Затем среднее время работы увеличивается при 5 потоках. При последующем увеличении числа потоков среднее время работы уменьшается;
- минимальное время работы алгоритма происходит на 4 потоках во всех случаях;

Стоит отметить, что при реализации случаев, когда сложность алгоритма равнялась  $O(1)$ , линейный алгоритм показывал результаты на несколько порядков лучше, чем параллельный:

mode	линейная	параллельная
0	$1.7 * 10^{-7}$ [с]	$4.915 * 10^{-3}$ [с]
4	$2.3 * 10^{-7}$ [с]	$4.946 * 10^{-3}$ [с]

Это связано с техническими ограничениями. Линейный алгоритм сразу после нахождения элемента может прервать выполнение цикла, а параллельный обязан пройти по всем итерациям.

Динамика значения среднего времени работы до 4 потоков во всех испытаниях, кроме второго, близко к теоретическим значениям. Отклонение от теоретического графика связано с потреблением временных ресурсов при создании новых потоков, а также при синхронизации на выходе из параллельного блока. Небольшой скачок при 5 потоках объясняется превышением требуемым числом потоков числа логических ядер.

Отклонением от теоритических значений наблюдается во втором испытании (**mode 1**), при котором при использовании **2 threads** экспериментальное время уменьшается больше чем в два раза, обганяя теоритическое значение. При **3 threads** время чуть увеличивается, возвращаясь к ожидаемому значению. Это связано с тем, что при **2 threads** элемент, находящийся посередине массива, оказывается в начале второго потока, из-за чего он находится сразу.

Для подтверждения сравним время **mode 0** и **mode 1** при **2 threads**:

threads	mode 0	mode 1
2	0.007029 [с]	0.007039 [с]

Значения различаются в десятых долях процента, что подтверждает предположение.

## 6. Приложение

Код программы расположен на github

Запуск программы: **./run**