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

ОТЧЁТ

Лабораторная работа №3:
«Реализация алгоритма с использованием технологии OpenMP»

Группа	Б21-525
Студент	Р.Т. Мясников
Преподаватель	М.А. Куприяшин

Москва 2023

Оглавление

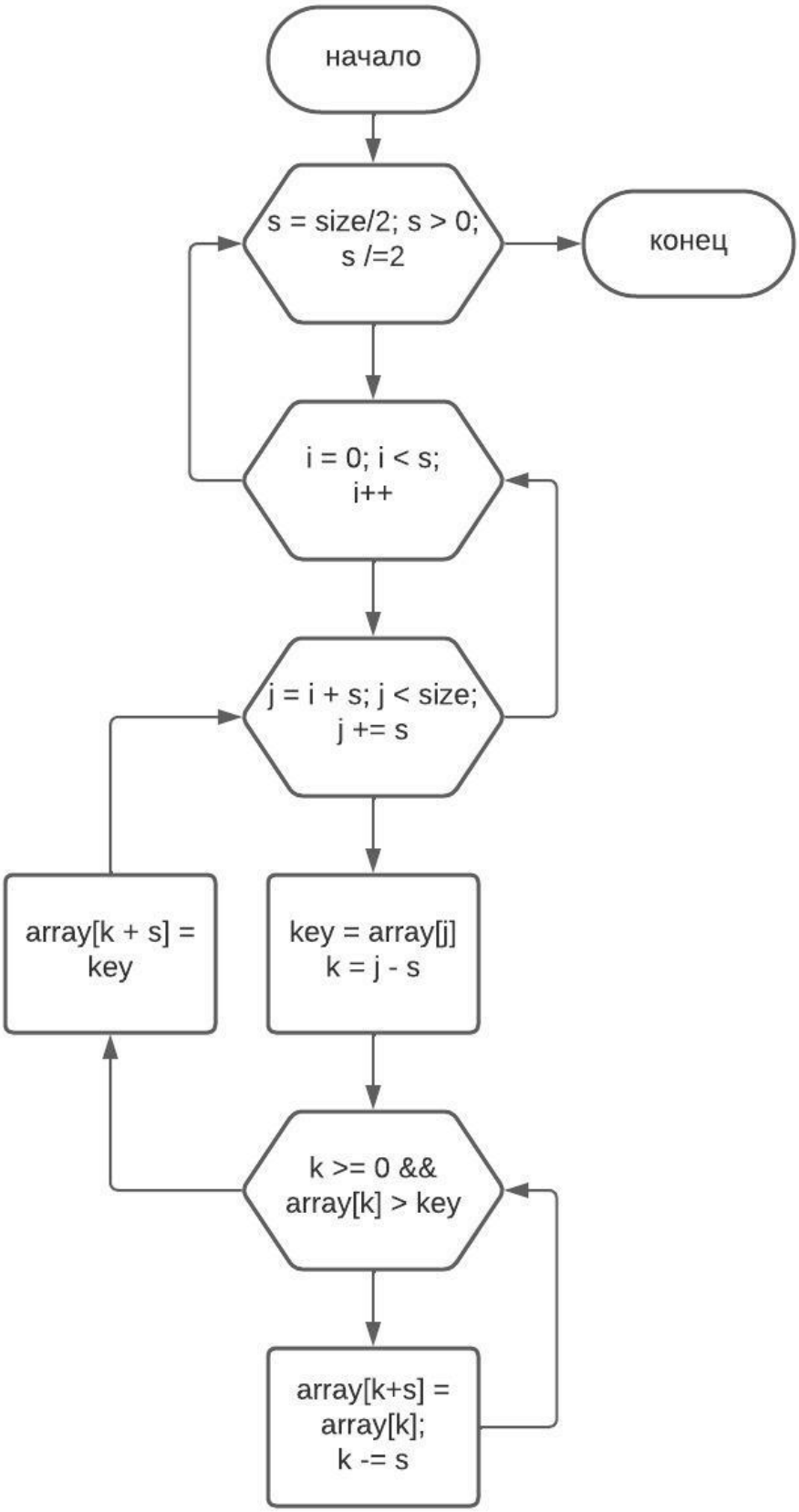
1. Описание рабочей среды

- Модель процессора:	Intel Core i3-10110U CPU @ 2.10GHz
- Число ядер:	2
- Число потоков:	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 `shell sort` массива.

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 for` распространяется на следующий блок программного кода:

```
#pragma omp parallel for num_threads(num_threads)
                         shared(array, size, s)
for (int i = 0; i < s; i++) {
    for (int j = i + s; j < size; j += s) {
        int key = array[j];
        int k = j - s;
        while (k >= 0 && array[k] > key) {
            array[k + s] = array[k];
            k -= s;
        }
        array[k + s] = key;
    }
}
```

Параметр `s` проходит от `size/2` до `1` уменьшаясь каждый раз в два раза. Такая реализация `shell sort` позволяет разделить массив между потоками так, чтобы их вычисления не пересекались. Однако, при малых значениях `s` (меньших числа потоков) не все потоки задействуются.

Описание работы алгоритма

Внешний цикл по `s` проходит от `size/2` до `1` уменьшаясь каждый раз в два раза. `s` задаёт растояние между элементами. Дерективой `parallel` объявляется блок кода, который будет исполняться параллельно. Далее внутри `for` потоки распередяют меду собой итерации цикла, отвечающие за подмассивы, элементы между которыми находятся на расстоянии `s`. Каждый поток проходит по своей части массива, сортируя элементы.

3. Анализ временных характеристик последовательного алгоритма

Описание эксперимента

- Эксперименты проводились на шести типах массивов

  mode 0: Рандомный массив
  mode 1: Отсортированный массив
  mode 2: Обратно упорядоченный массив
  mode 3: Все элементы одинаковые
  mode 4: Каждый элемент встречается в среднем 10 000 раз
  mode 5: Каждый элемент встречается в среднем 100 раз
  mode 6: Замиксованные отсортированный и обратно упорядоченный массивы

- измеряется время работы алгоритма на 100 различных массивах длиной 1 000 000 элементов. Находится среднее значение;

Экспериментальные показатели

- Среднее время работы последовательного алгоритма

  mode 0: 0.285433 [c]
  mode 1: 0.092019 [c]
  mode 2: 0.105432 [c]
  mode 3: 0.082512 [c]
  mode 4: 0.186740 [c]
  mode 5: 0.319439 [c]
  mode 6: 0.093396 [c]

4. Анализ временных характеристик параллельного алгоритма

Описание эксперимента

- Эксперименты проводились на шести типах массивов

  mode 0: Рандомный массив
  mode 1: Отсортированный массив
  mode 2: Обратно упорядоченный массив
  mode 3: Все элементы одинаковые
  mode 4: Каждый элемент встречается в среднем 10 000 раз
  mode 5: Каждый элемент встречается в среднем 100 раз
  mode 6: Замиксованные отсортированный и обратно упорядоченный массивы

- измеряется время работы алгоритма для одного и того же массива, но на разном числе потоков: от 1 до 10;
- измеряется время работы алгоритма на 100 различных массивах длиной 1 000 000 элементов. Находится среднее значение;

Результаты измерений

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

threads	mode 0
1	0.285130
2	0.179470
3	0.165701
4	0.137576
5	0.158755
6	0.152433
7	0.146690
8	0.139548
9	0.149686
10	0.147449

threads	mode 1
1	0.079944
2	0.059001
3	0.057136
4	0.050169
5	0.054587
6	0.054189
7	0.053246
8	0.050888
9	0.053030
10	0.052987

threads	mode 2
1	0.107072
2	0.076465
3	0.071793
4	0.067846
5	0.068699
6	0.069600
7	0.067549
8	0.063990
9	0.067032
10	0.066974

threads	mode 3
1	0.077978
2	0.059551
3	0.057636
4	0.050111
5	0.053542
6	0.053611
7	0.053748
8	0.050474
9	0.052587
10	0.052957

threads	mode 4
1	0.157966
2	0.103146
3	0.093285
4	0.078245
5	0.091913
6	0.089011
7	0.084712
8	0.079088
9	0.085387
10	0.084243

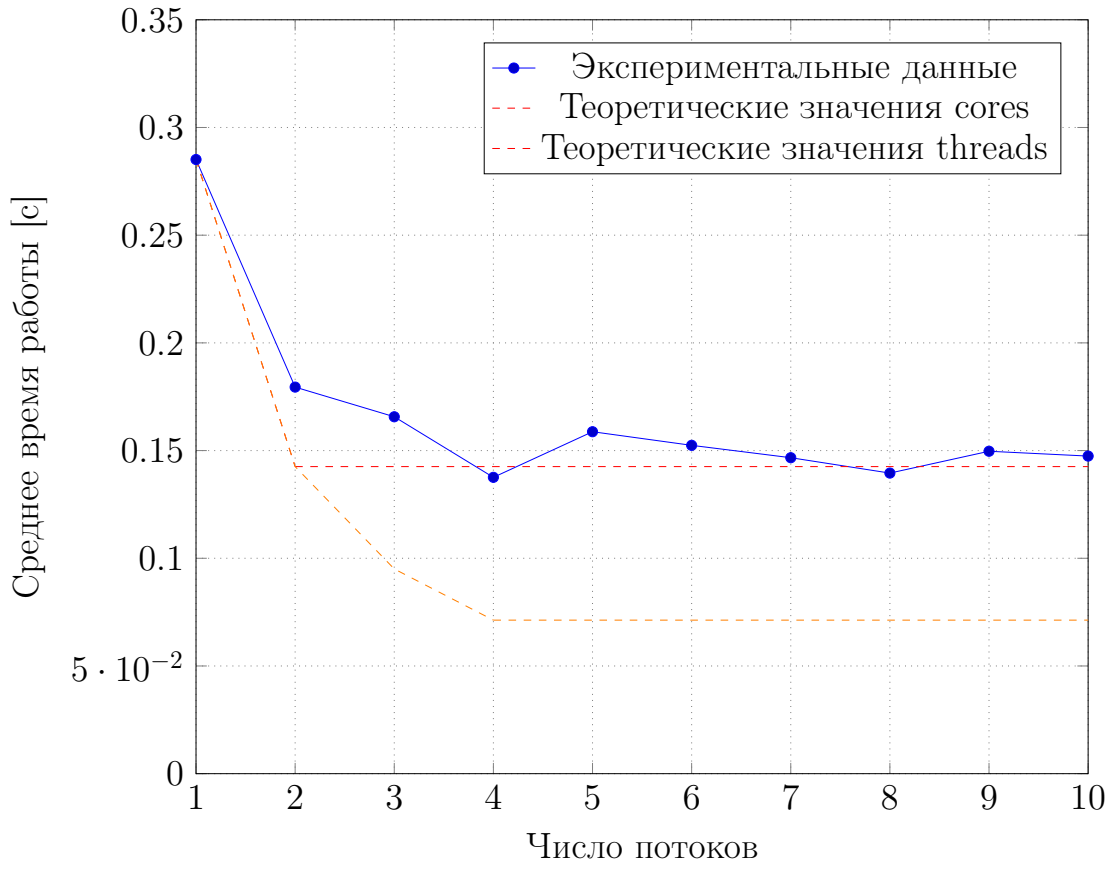
threads	mode 5
1	0.272008
2	0.165607
3	0.154502
4	0.131102
5	0.149741
6	0.141975
7	0.137015
8	0.130159
9	0.143385
10	0.140698

threads	mode 6
1	0.099367
2	0.069918
3	0.067955
4	0.057575
5	0.063742
6	0.064246
7	0.062627
8	0.058811
9	0.062226
10	0.061474

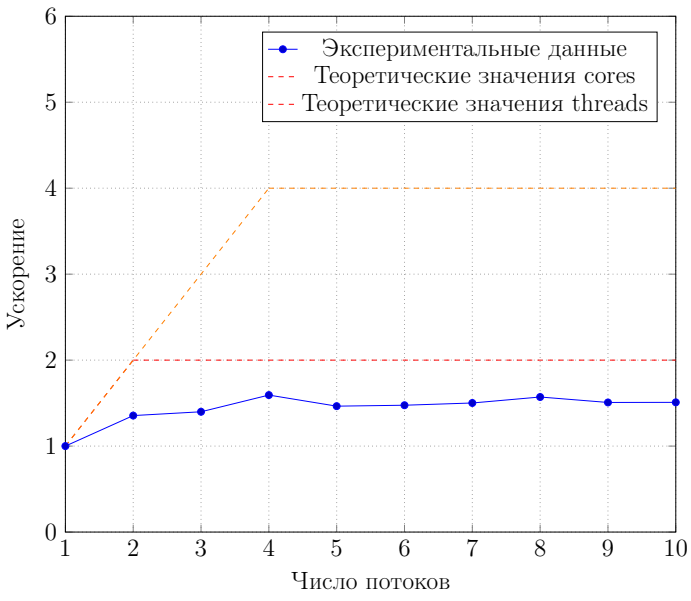
Графики

1. Рандомный массив (mode 0)

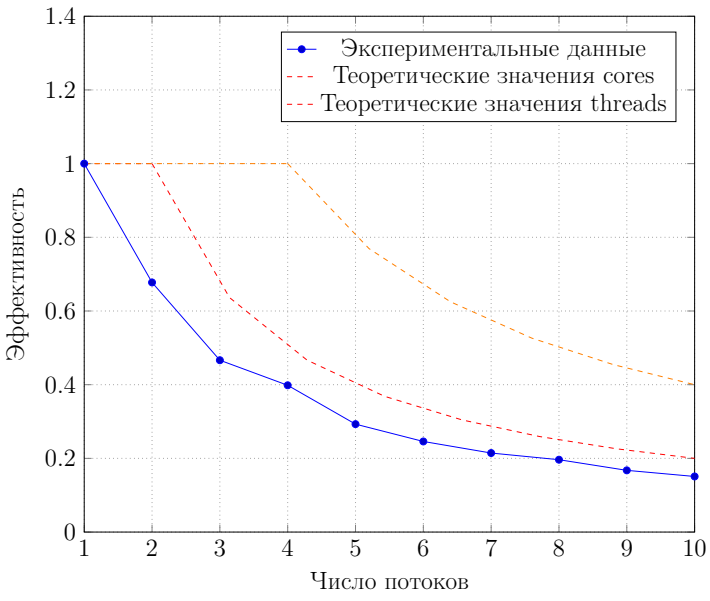
Зависимость времени работы от числа потоков



Зависимость ускорения от числа потоков

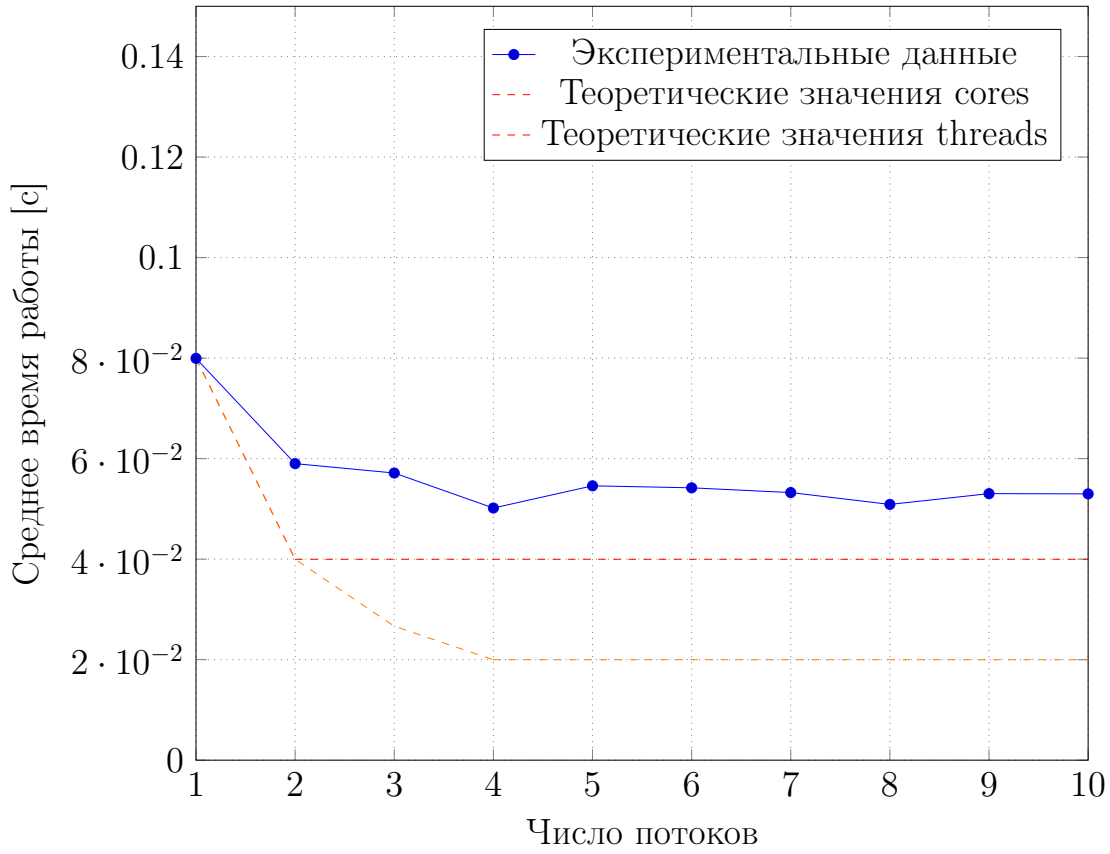


Зависимость эффективности от числа потоков

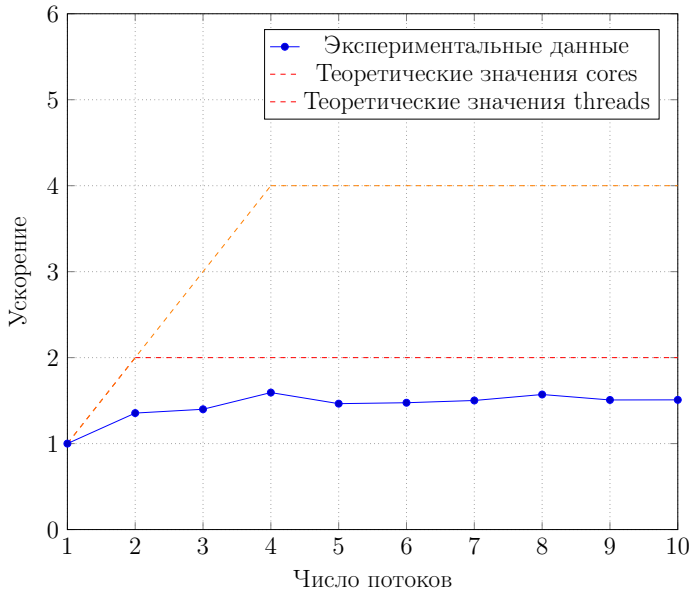


2. Отсортированный массив (mode 1)

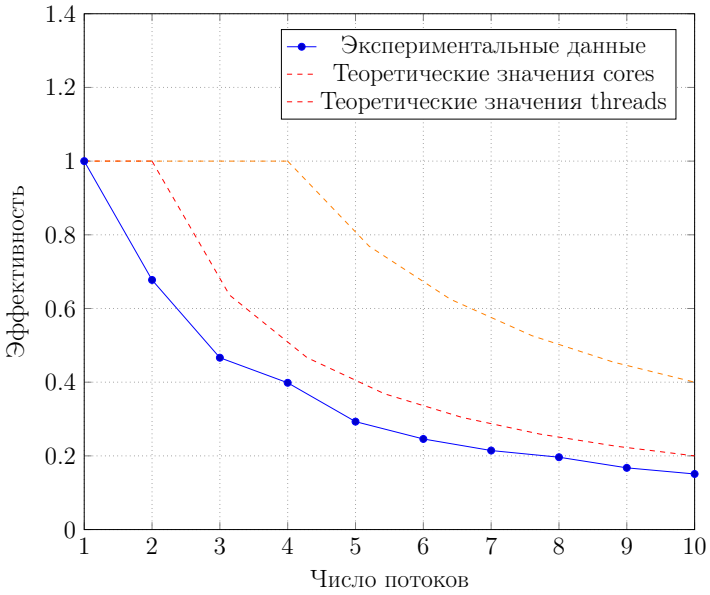
Зависимость времени работы от числа потоков



Зависимость ускорения от числа потоков

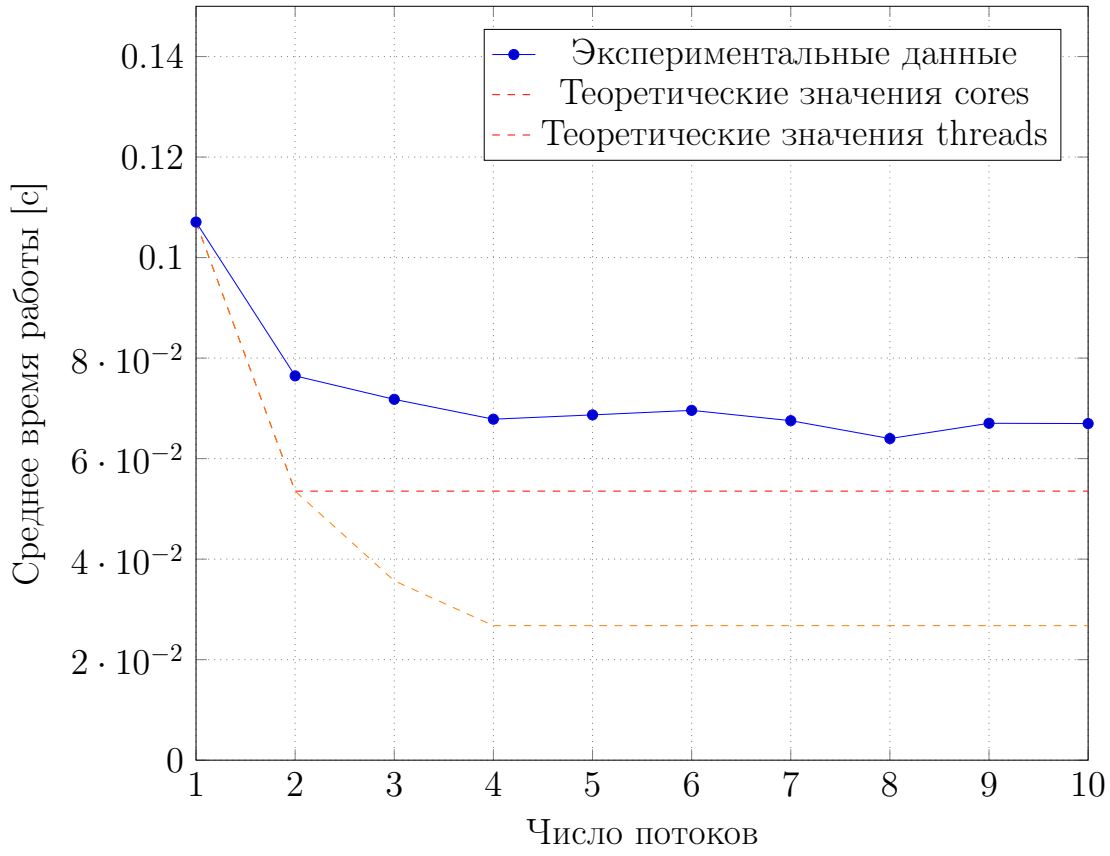


Зависимость эффективности от числа потоков

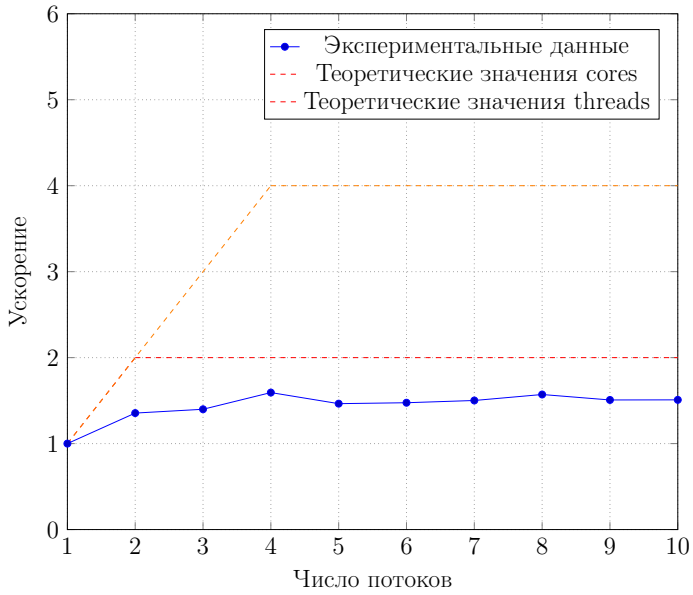


3. Обратно упорядоченный массив (mode 2)

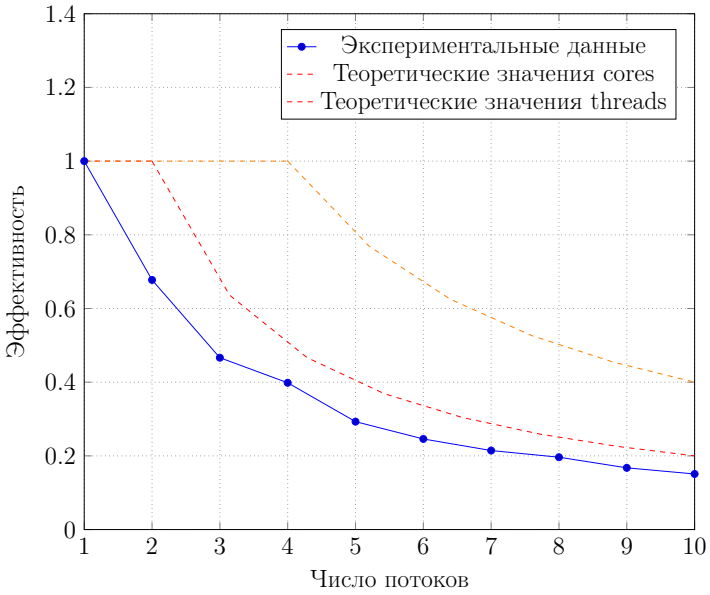
Зависимость времени работы от числа потоков



Зависимость ускорения от числа потоков

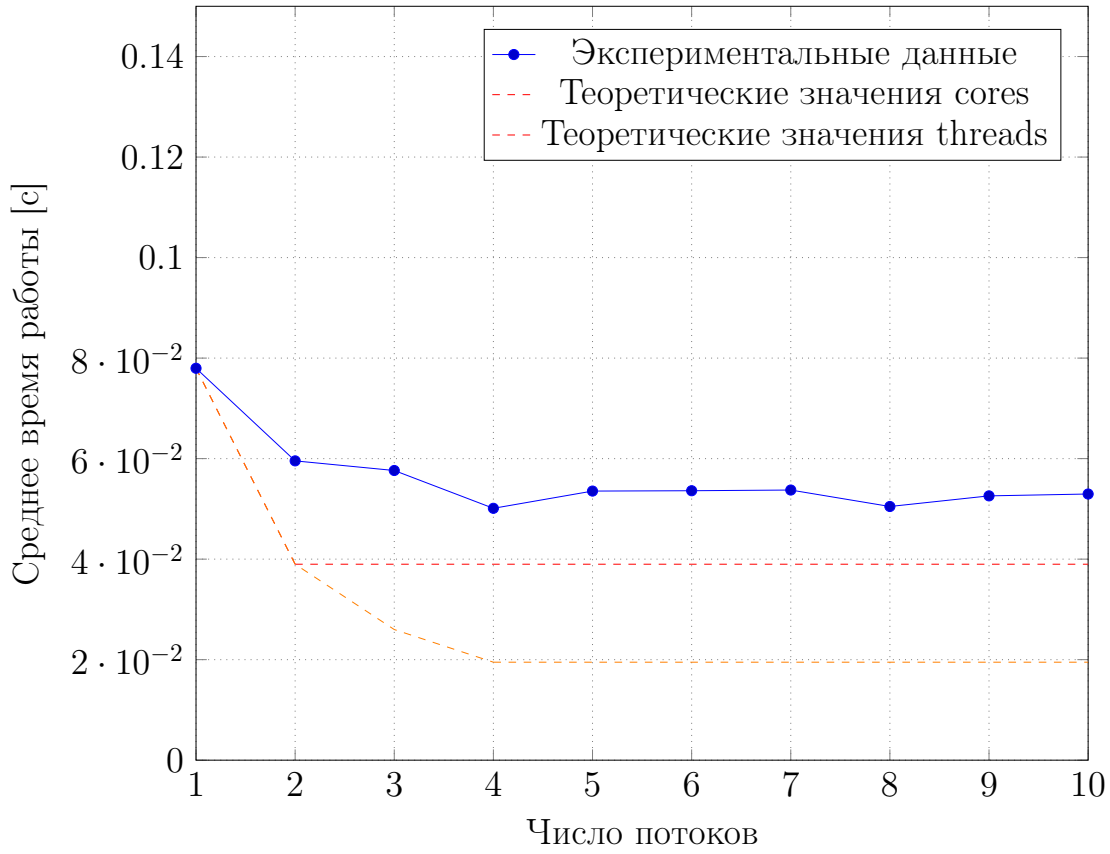


Зависимость эффективности от числа потоков

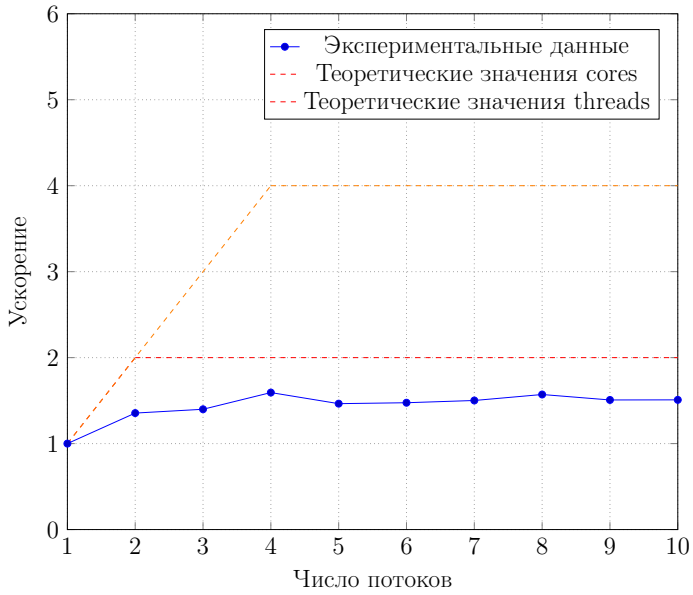


4. Все элементы одинаковые (mode 3)

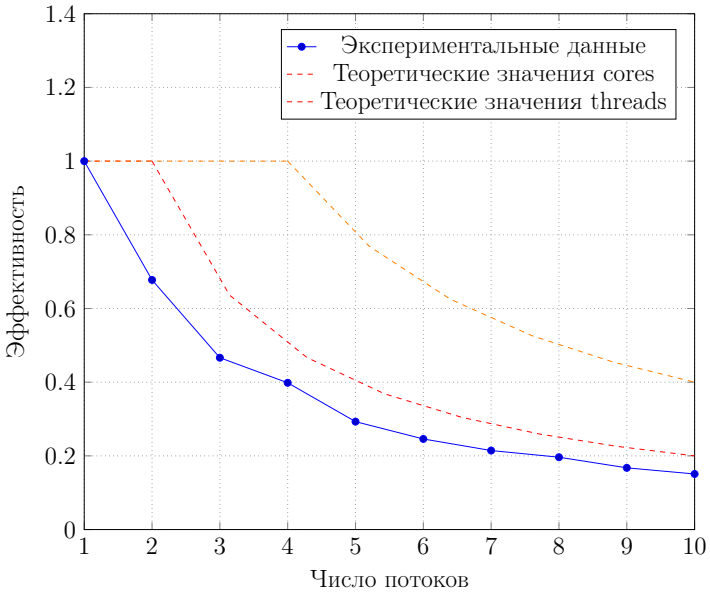
Зависимость времени работы от числа потоков



Зависимость ускорения от числа потоков

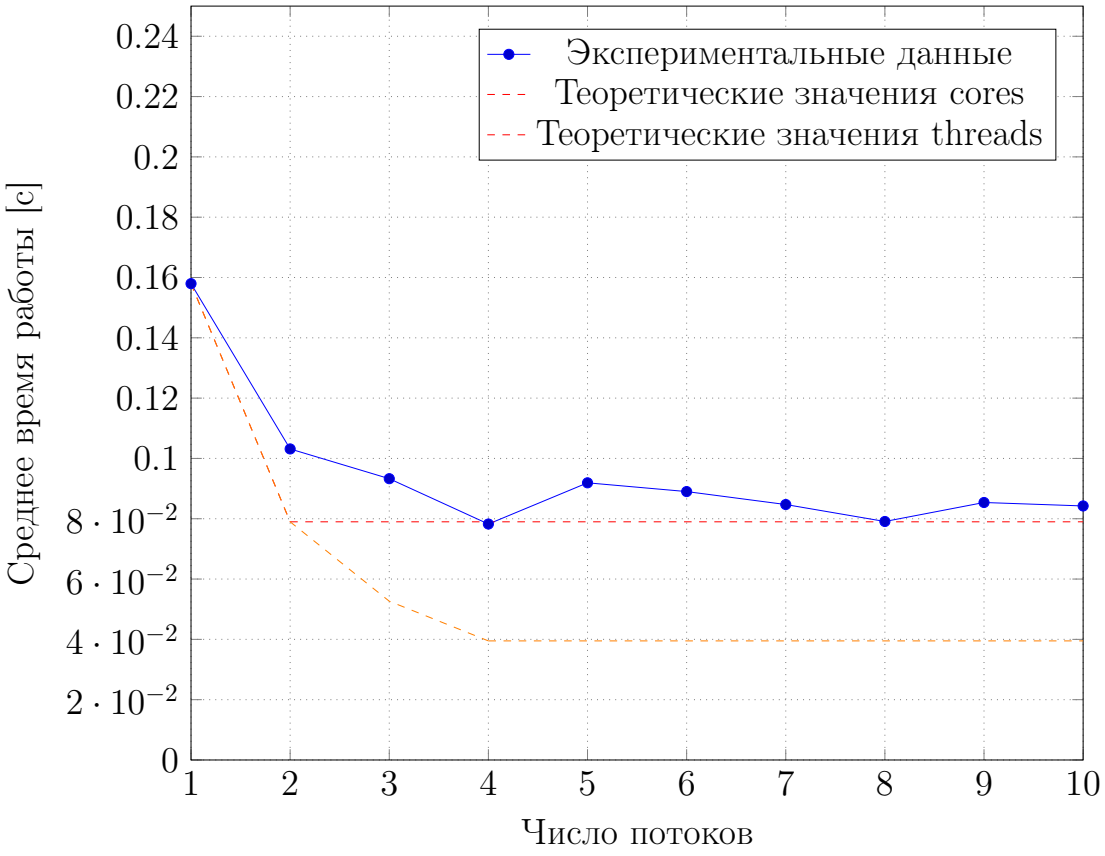


Зависимость эффективности от числа потоков

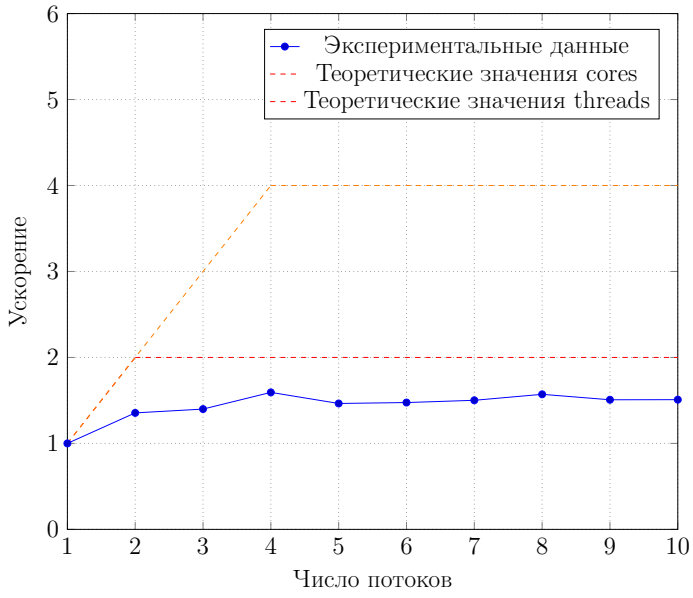


5. Каждый элемент встречается в среднем 10 000 раз (mode 4)

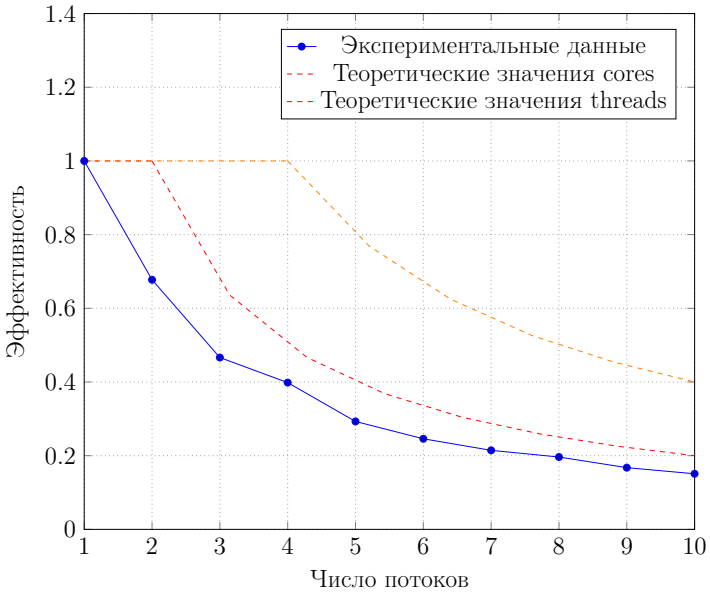
Зависимость времени работы от числа потоков



Зависимость ускорения от числа потоков

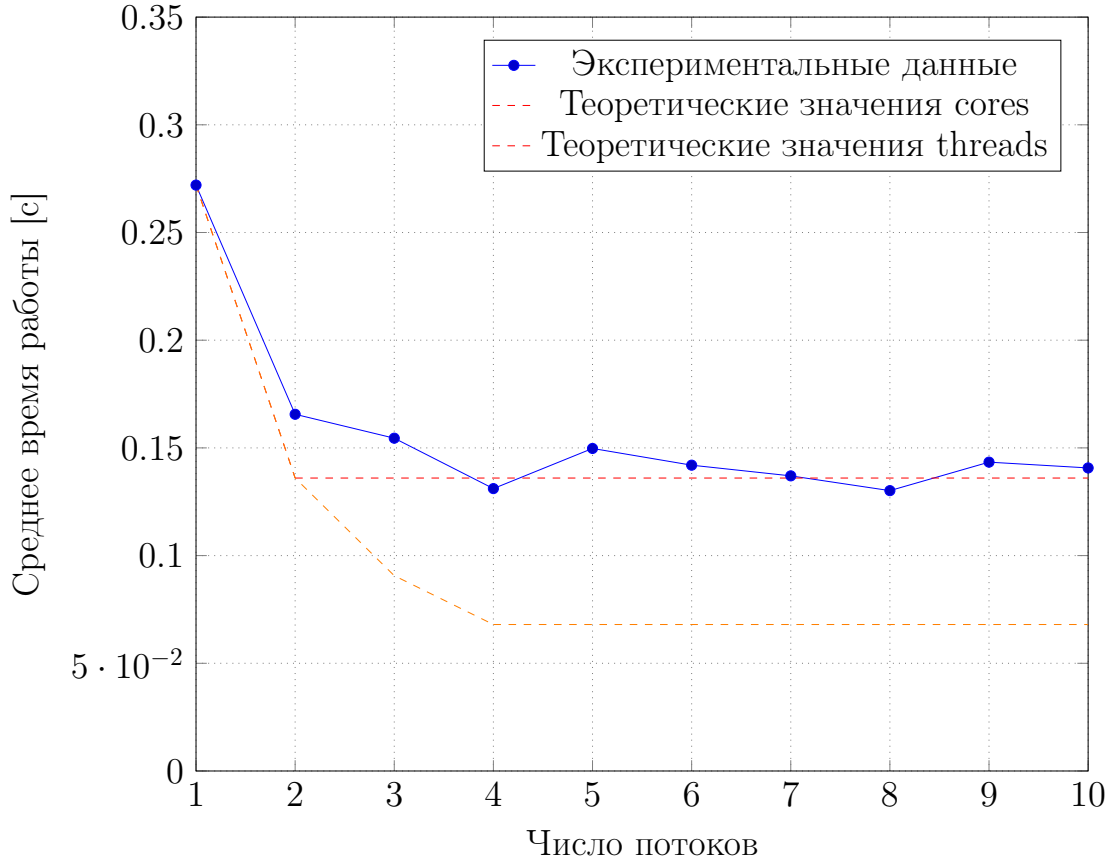


Зависимость эффективности от числа потоков

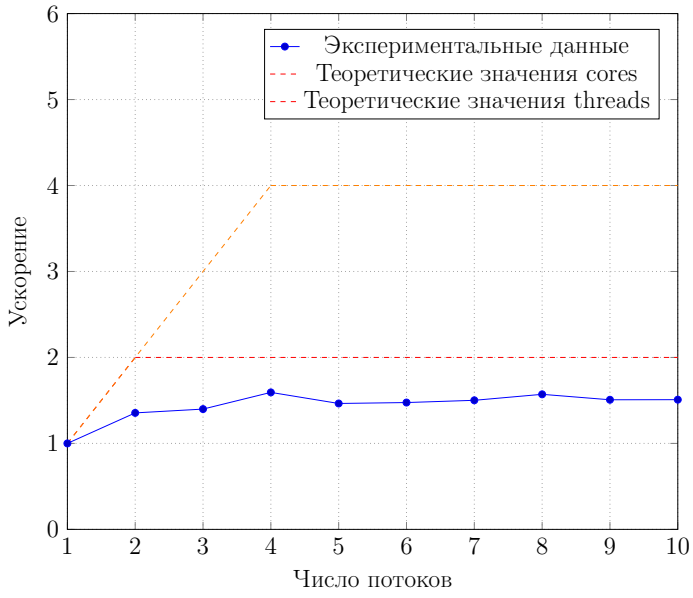


6. Каждый элемент встречается в среднем 100 раз (mode 5)

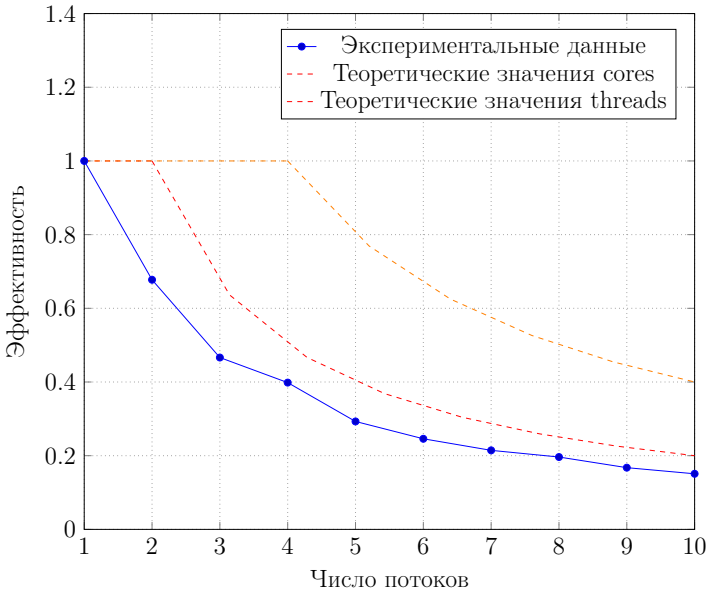
Зависимость времени работы от числа потоков



Зависимость ускорения от числа потоков

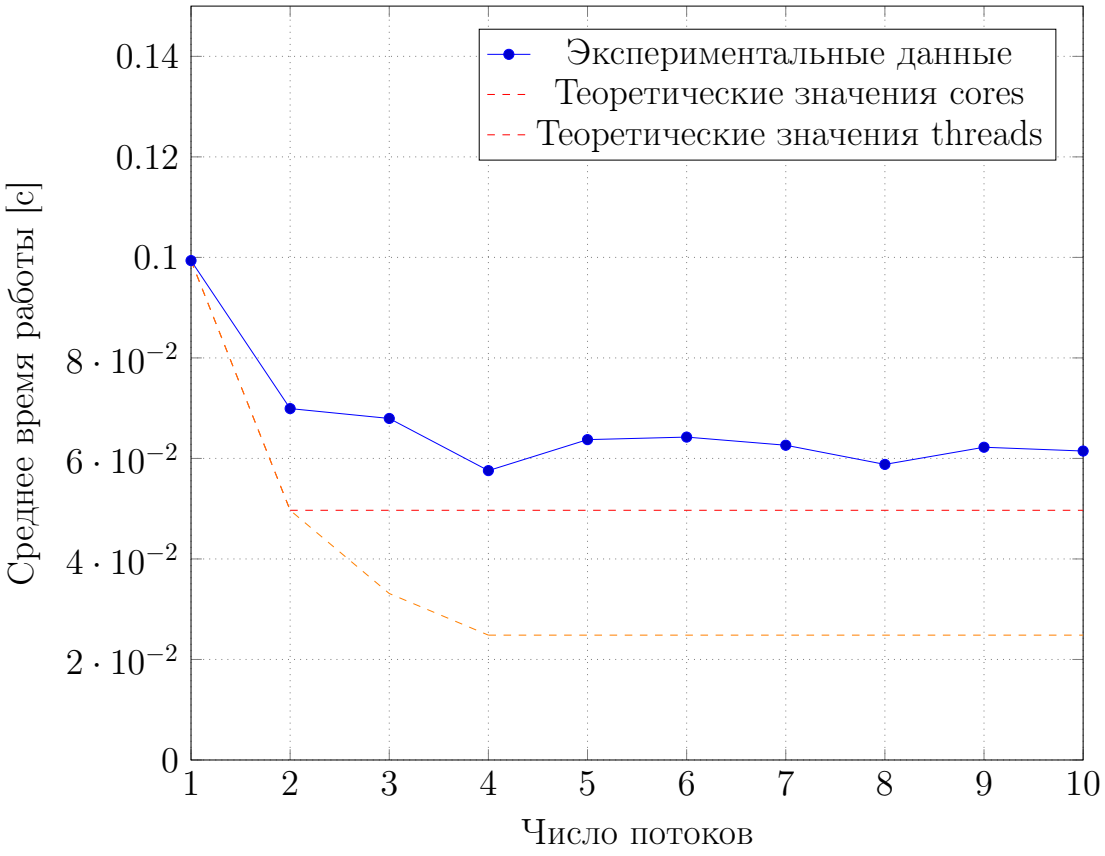


Зависимость эффективности от числа потоков

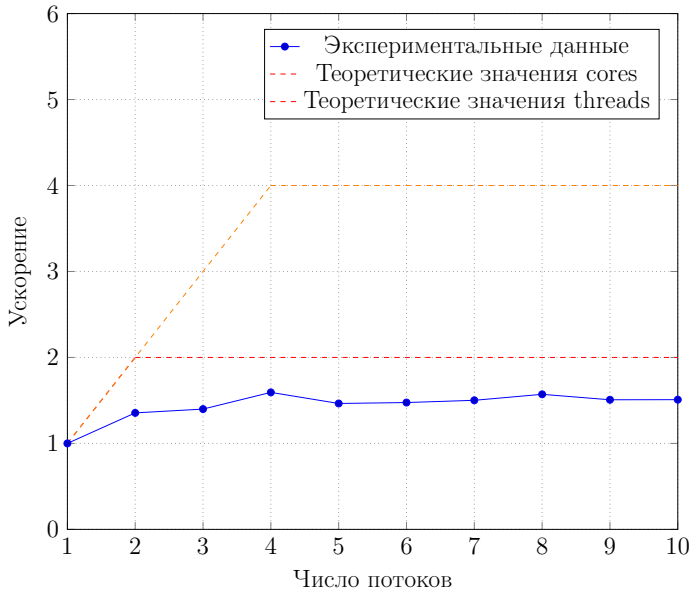


7. Замиксованные отсортированный и обратно упорядоченный массивы (mode 6)

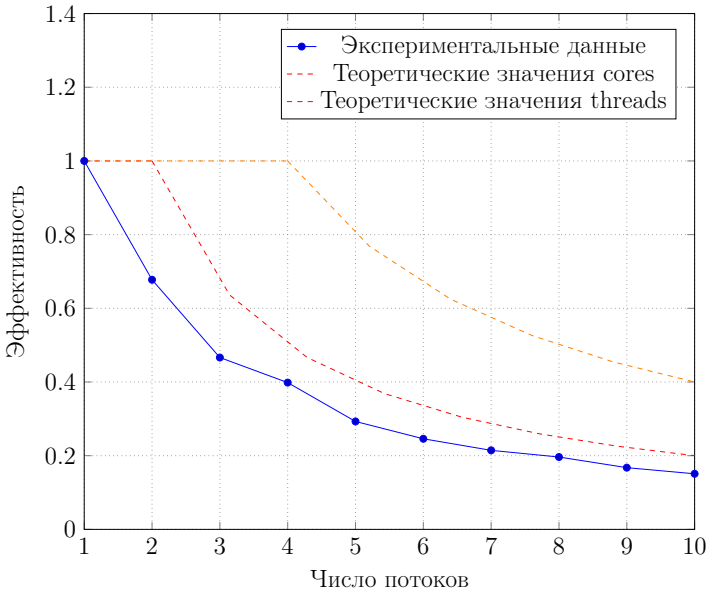
Зависимость времени работы от числа потоков



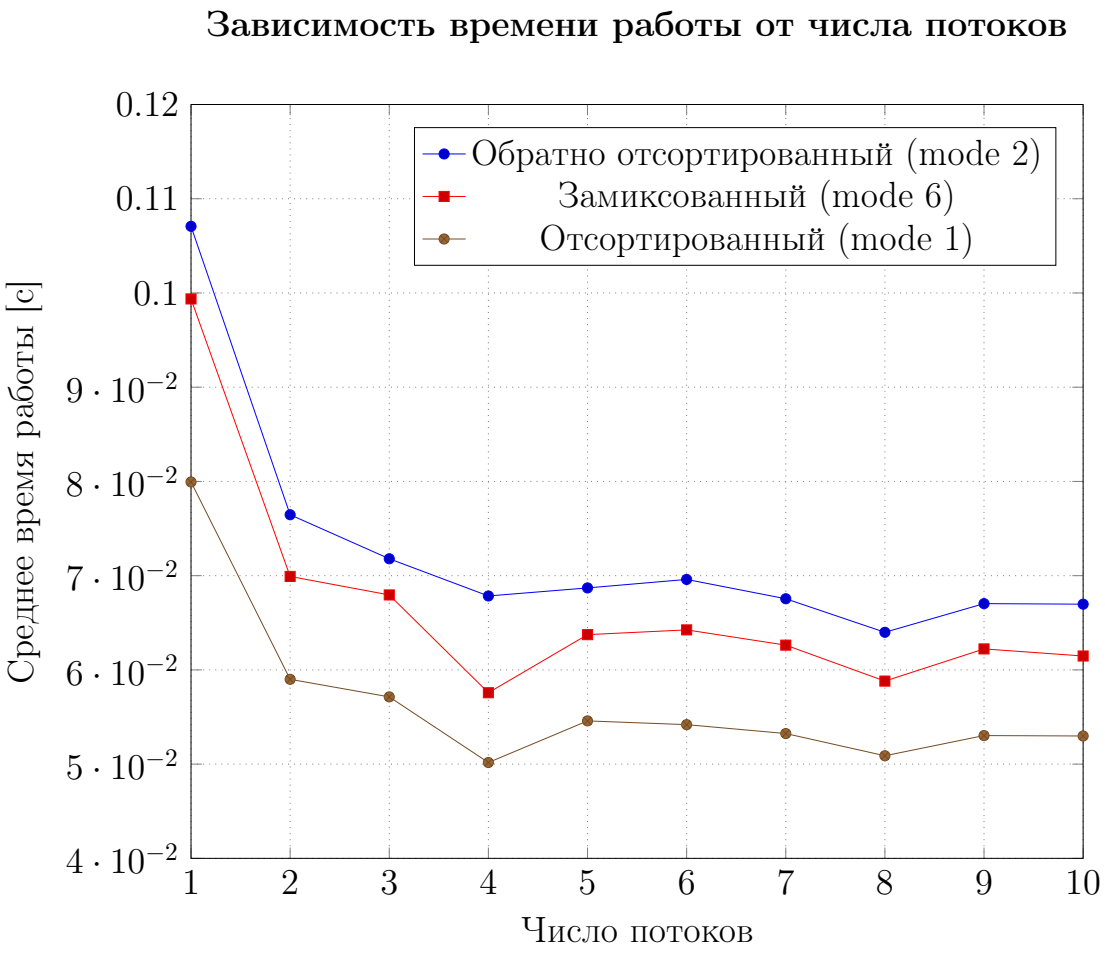
Зависимость ускорения от числа потоков



Зависимость эффективности от числа потоков



8. Сравнение отсортированного, обратно упорядоченного и замиксованного массива



5. Заключение

В ходе лабораторной работы был разработан алгоритм `shell sort` на языке Си, допускающий использования нескольких потоков путём разделения массива на подмассивы. Было измерено время работы алгоритма `shell sort` массива 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:

- наибольшая `скорость` была получена при сортировки массивы из одинаковых элементов `(mode 3)` при `4 threads`: `0.050111 [c]`;
- наибольшее `ускорение` было получено при сортировки массива, где каждый элемент встречается в среднем 100 раз `(mode 5)` при `4 threads`: `2.074781`;
- наибольшая `эффективность` была получена при сортировки массива, где каждый элемент всречается в среднем 100 раз `(mode 5)` при `2 threads`: `0.821246`;
- минимальное время работы алгоритма происходило на `4 threads` во всех случиях, кроме `(mode 2)` и `(mode 5)`;
- минимальное время работы алгоритма для `(mode 2)` происходило на `8 threads`; сравнивая с временем выполнени на `4 threads` ускорение состовило `6 %`
- минимальное время работы алгоритма для `(mode 5)` происходило на `8 threads`; сравнивая с временем выполнени на `4 threads` ускорение состовило `1 %`
- экспериментальные графики близки к теоритическим графикам по значению `cores`, показывая хорошее ускорение при увеличении до `2 threads`
- средний прирост времени относительно линейного алгоритма в `1.99 раз`
- график времени выполнения сортировки замиксованного массива `(mode 6)` лежит между графиками отсортированного `(mode 1)` и обратно отсортированного `(mode 2)`

6. Приложение

Код программы расположен на github
Запуск программы: `./run`