

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ
«МИФИ»
КАФЕДРА №42 «КРИПТОЛОГИЯ И КИБЕРБЕЗОПАСНОСТЬ»

ОТЧЁТ

по дисциплине «Параллельное программирование»
Лабораторная работа №3
«Реализация алгоритма с использованием технологии
OpenMP»

Группа

Б21-525

Студент

Г.О. Шулындин

Преподаватель

М.А. Куприяшин

Москва 2023

Оглавление

1.	Описание рабочей среды	3
2.	Анализ линейного алгоритма	3
3.	Построение параллельного алгоритма	5
4.	Анализ временных характеристик параллельного алгоритма	6
5.	Заключение	16
6.	Приложение	16

1. Описание рабочей среды

- Модель процессора: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
- Число ядер: 8
- Архитектура: x86-64
- ОС: Linux, дистрибутив Ubuntu v22.04
- RAM объем: 2x4096 MB
- RAM тип: DDR4
- Используемая среда разработки: Visual Studio Code
- Компилятор: gcc v11.4.0
- Поддерживаемая версия OpenMP: 201511

2. Анализ линейного алгоритма

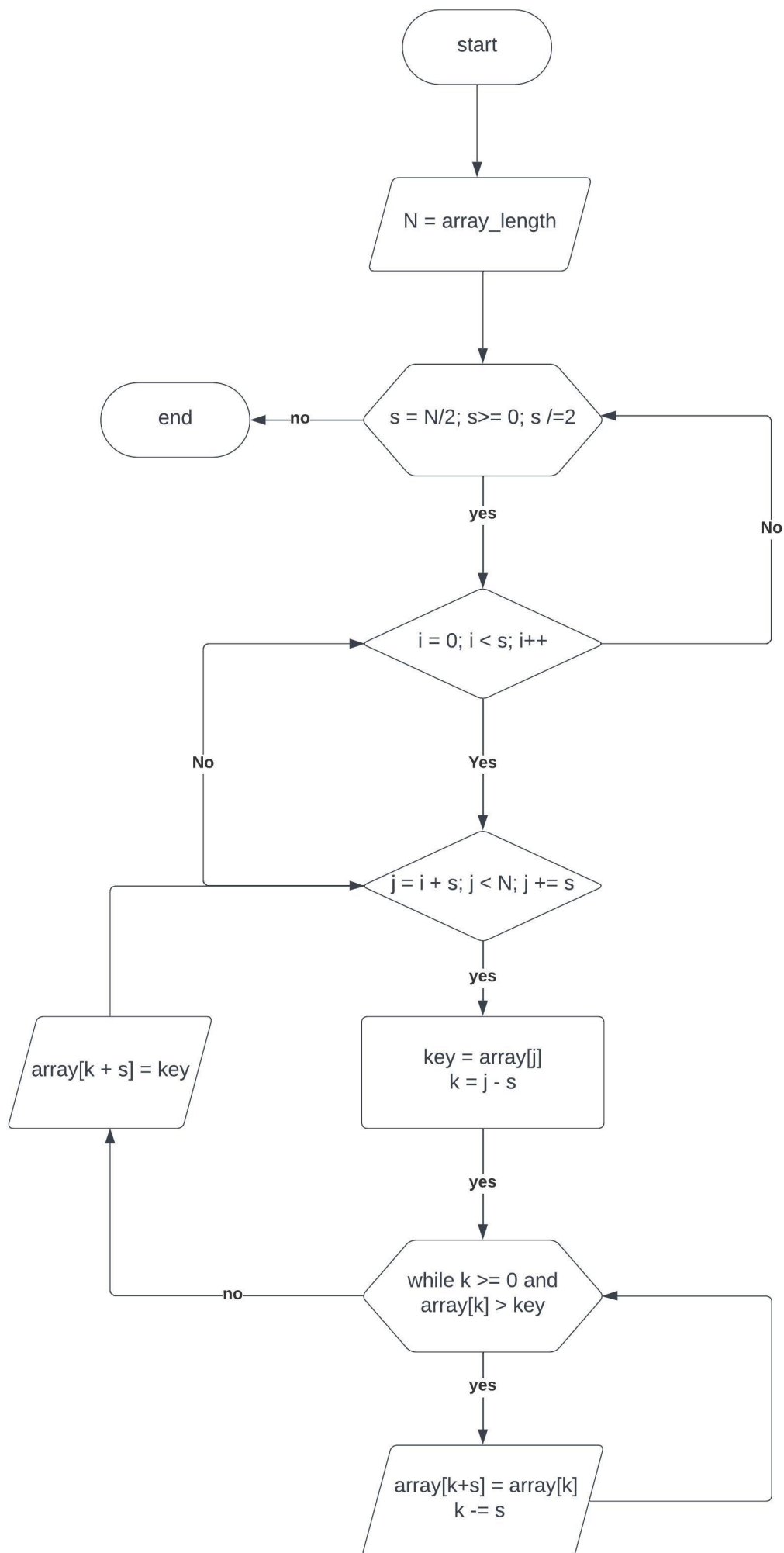
Принцип работы алгоритма

Пусть массив имеет длину N . В данной реализации сортировки Шелла, длина промежутков s_k вычисляется, как $s_k = \frac{N}{2^k}$. То есть сначала происходит сортировка вставками подмассивов элементов, находящихся на расстоянии $N/2$, затем - сортировка вставками подмассивов элементов, находящихся на расстоянии $N/4$ и так далее. В конце сортируется массив целиком. Преимущество сортировки Шелла перед сортировкой вставками состоит в том, что на начальных этапах сравниваются и обмениваются элементы на большом расстоянии. Это позволяет стремительнее уменьшать количество инверсий.

Ниже приведена реализация линейного алгоритма сортировки Шелла, используемая в данной работе:

```
1 double shell_sort_linear(int *array, int array_length) {
2     ....
3     for (int s = array_length / 2; s > 0; s /= 2) {
4         for (int i = 0; i < s; i++) {
5             for (int j = i + s; j < array_length; j += s) {
6                 int key = array[j];
7                 int k = j - s;
8                 while (k >= 0 && array[k] > key) {
9                     array[k + s] = array[k];
10                    k -= s;
11                }
12                array[k + s] = key;
13            }
14        }
15    }
16    ....
17 }
```

Блоксхема алгоритма



3. Построение параллельного алгоритма

Выявление области распараллеливания

В вышеприведенном алгоритме имеет смысл распараллеливания вложенного цикла `for`, так как в нем происходит разбиение исходного массива на s_k не пересекающихся подмассивов, внутри которых происходит сортировка вставками. Важно, что подмассивы не пересекающиеся, т.к. иначе при обмене элементов в подмассиве, сортируемом одним потоком, может нарушиться сортировка, выполненная на другом подмассиве другим потоком.

Реализация параллельного алгоритма

```
1 double shell_sort_n_threads(int *array, int array_length, int
   num_threads) {
2     int s, i;
3     ....
4     for (s = array_length / 2; s > 0; s /= 2) {
5         #pragma omp parallel num_threads(num_threads) shared(array, s,
   array_length) private (i) default(none)
6         {
7             #pragma omp for
8             for (i = 0; i < s; i++) {
9                 for (int j = i + s; j < array_length; j += s) {
10                    int key = array[j];
11                    int k = j - s;
12                    while (k >= 0 && array[k] > key) {
13                        array[k + s] = array[k];
14                        k -= s;
15                    }
16                    array[k + s] = key;
17                }
18            }
19        }
20    }
21    ....
22 }
```

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;
- `shared()` - объявляет, что одна или несколько переменных должны быть общими между всеми потоками;
- `private()` - объявляет, что одна или несколько переменных должны быть общими между всеми потоками;
- `default()` - указывает поведение по умолчанию переменных в параллельной области;

`for` - приводит к разделу работы, выполняемой в цикле `for` внутри параллельной области, между потоками.

4. Анализ временных характеристик параллельного алгоритма

Описание эксперимента

Условия эксперимента:

- измеряется время работы алгоритма для одного и того же массива на разном числе потоков: от 1 до 10;
- измерения производятся для 10 различных массивов одного типа;
- размер каждого массива 100 000 элементов.

Были выделены следующие типы массивов:

- массив упорядочен (Type 0);
- массив обратно упорядочен (Type 1);
- случайный массив (Type 2);
- в массиве много одинаковых чисел (Type 3);
- первая половина массива упорядочена (Type 4);
- первая половина массива обратно упорядочена (Type 5);
- первая четверть массива упорядочена (Type 6);
- первая четверть массива обратно упорядочена (Type 7).

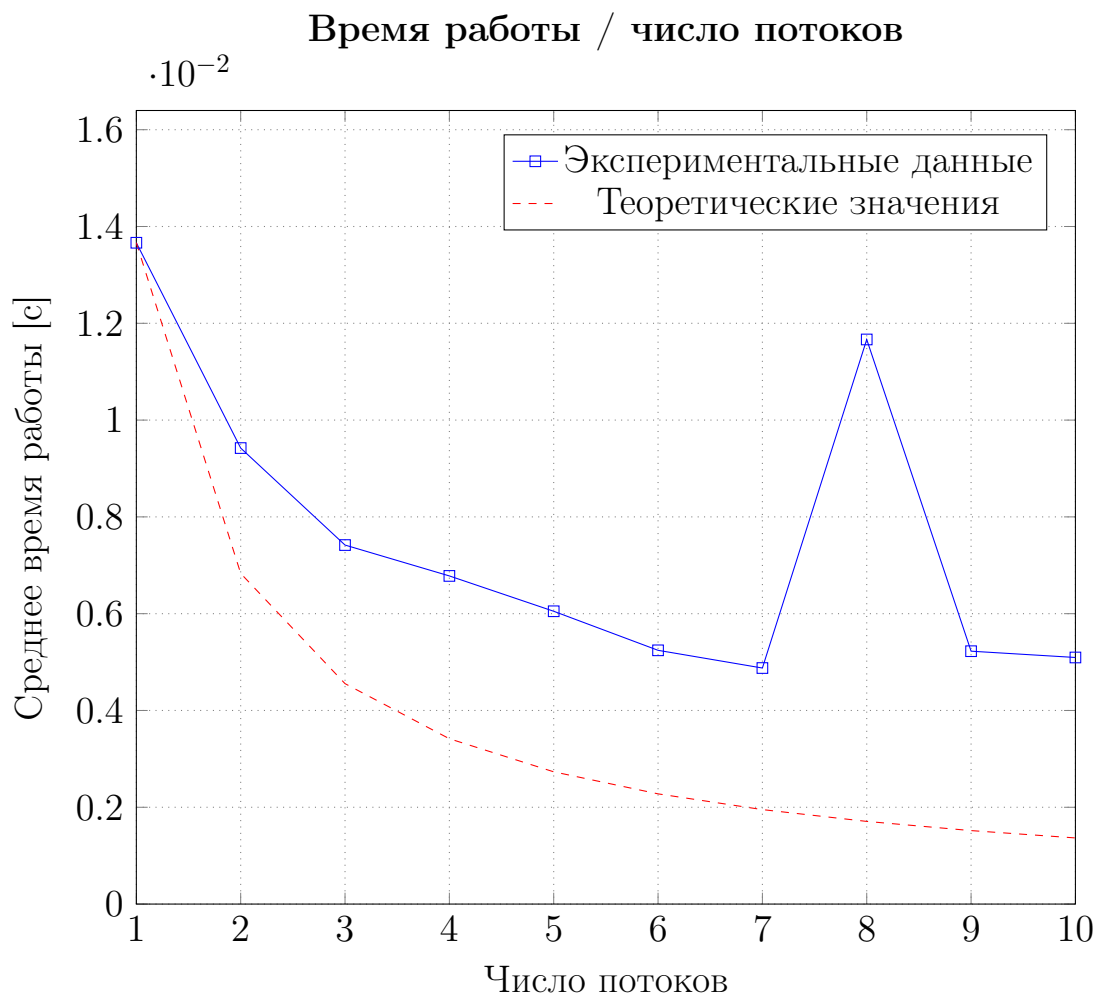
Результаты измерений

Следующая таблица содержит полученные в результате эксперимента данные: среднее время работы на различном числе потоков для массивов различных типов.

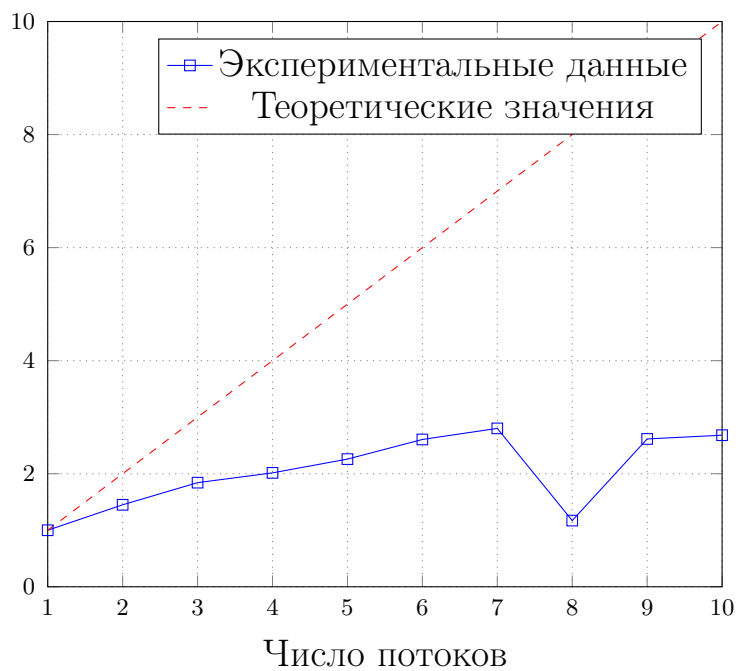
Thds num.	Type 0	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6	Type 7
1	0.013664	0.017902	0.037184	0.025148	0.035134	0.035081	0.036921	0.036119
2	0.009422	0.012437	0.030148	0.019784	0.028548	0.026604	0.031038	0.027465
3	0.007418	0.009598	0.026658	0.014832	0.023917	0.024291	0.025794	0.025742
4	0.00678	0.00811	0.020851	0.013081	0.020278	0.020354	0.020914	0.020819
5	0.00605	0.008717	0.018291	0.011917	0.018264	0.018148	0.017986	0.019367
6	0.005243	0.007698	0.015862	0.009782	0.01585	0.01626	0.015211	0.015396
7	0.004877	0.008372	0.01564	0.008714	0.014859	0.015717	0.014534	0.014913
8	0.011666	0.009116	0.016072	0.010192	0.016186	0.015062	0.014224	0.015142
9	0.005225	0.006996	0.013945	0.008304	0.013366	0.013507	0.013901	0.013758
10	0.005095	0.006784	0.013422	0.008413	0.012856	0.012991	0.014141	0.013667

Графики

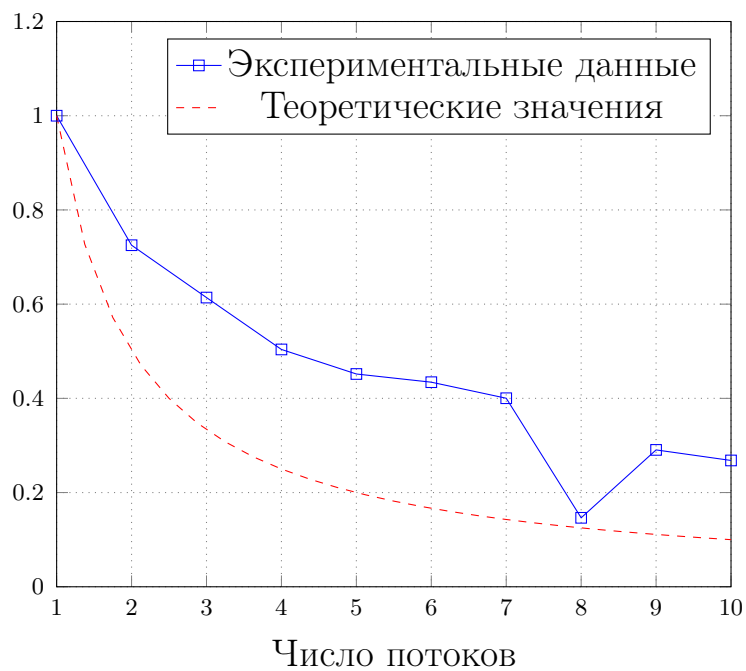
Массив упорядочен (Type 0)



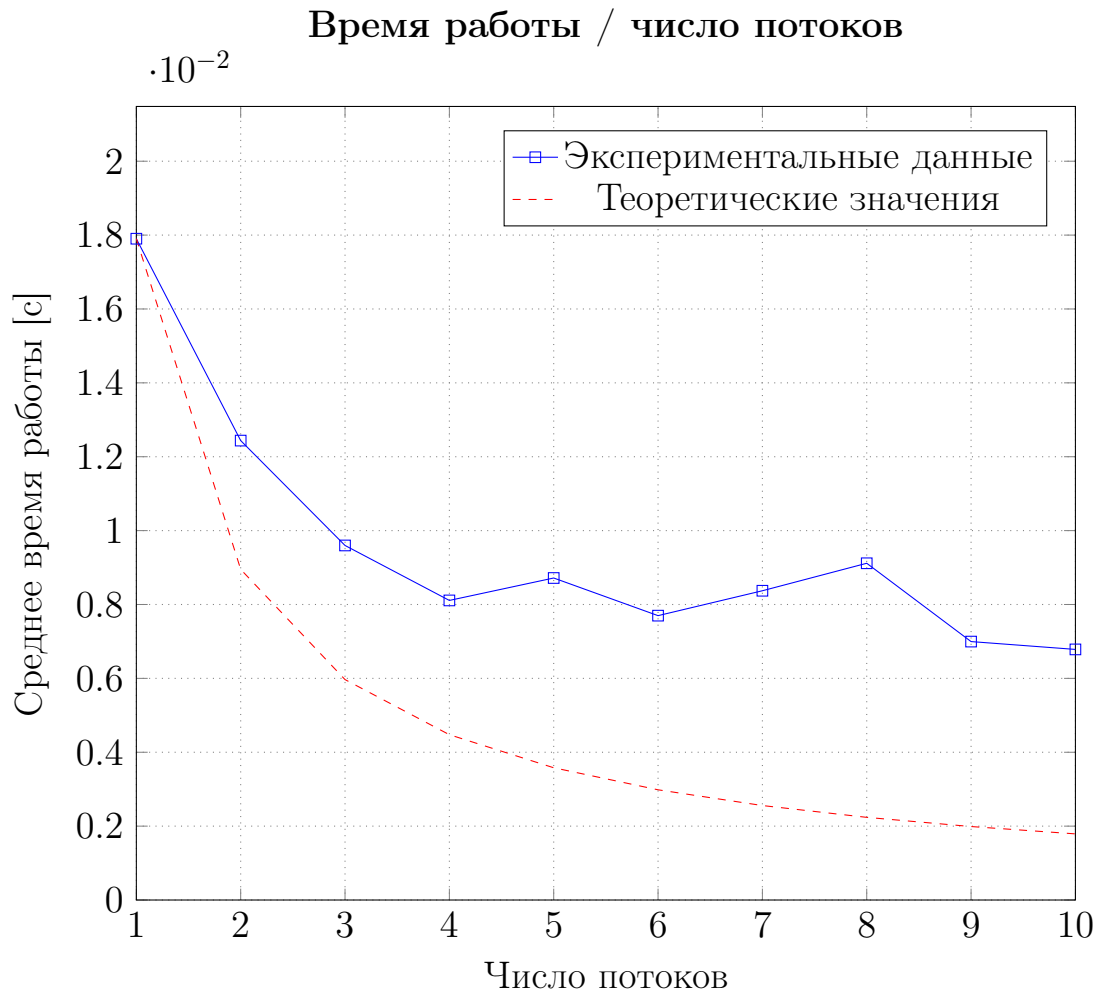
Ускорение / число потоков



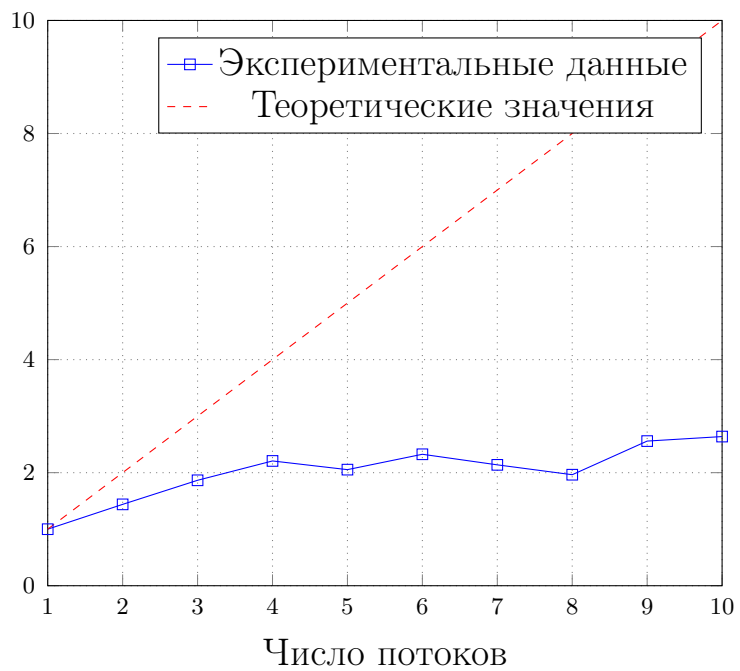
Эффективность / число потоков



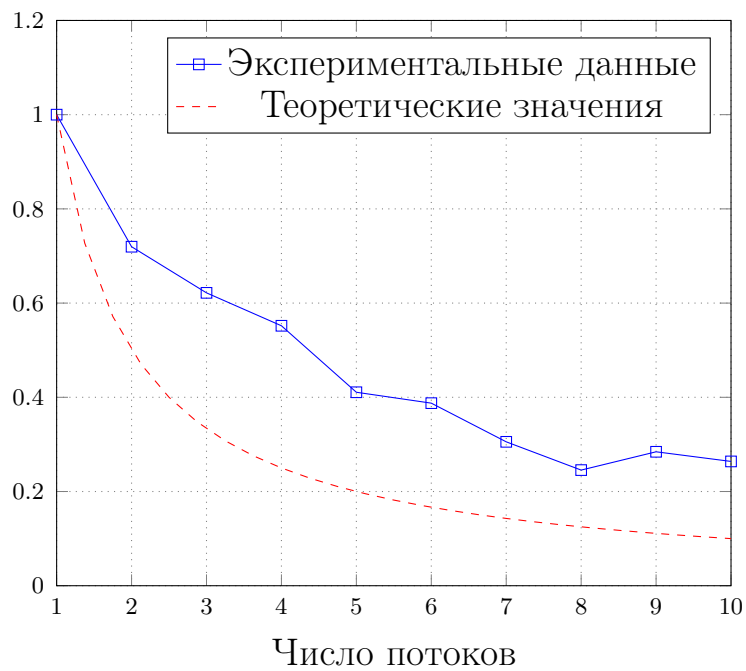
Массив обратно упорядочен (Type 1)



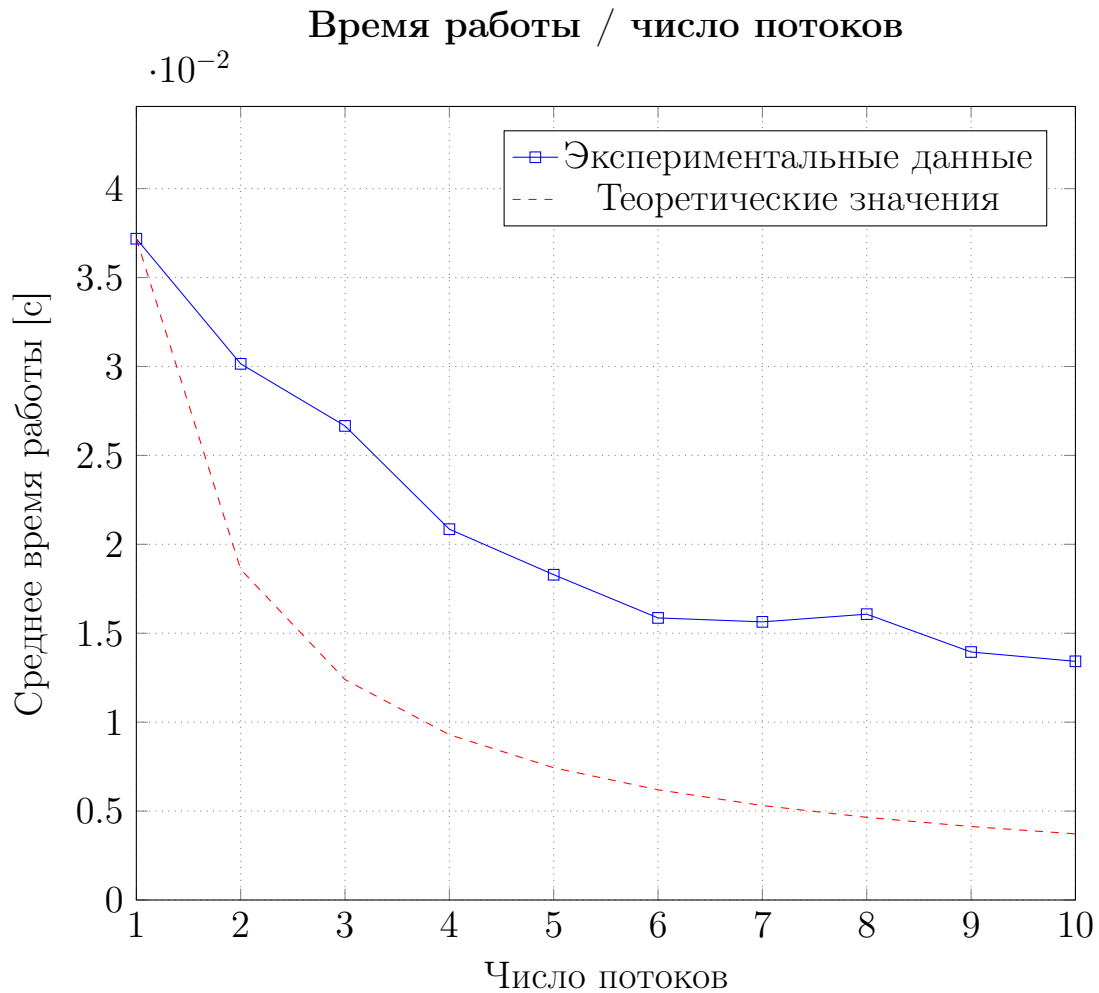
Ускорение / число потоков



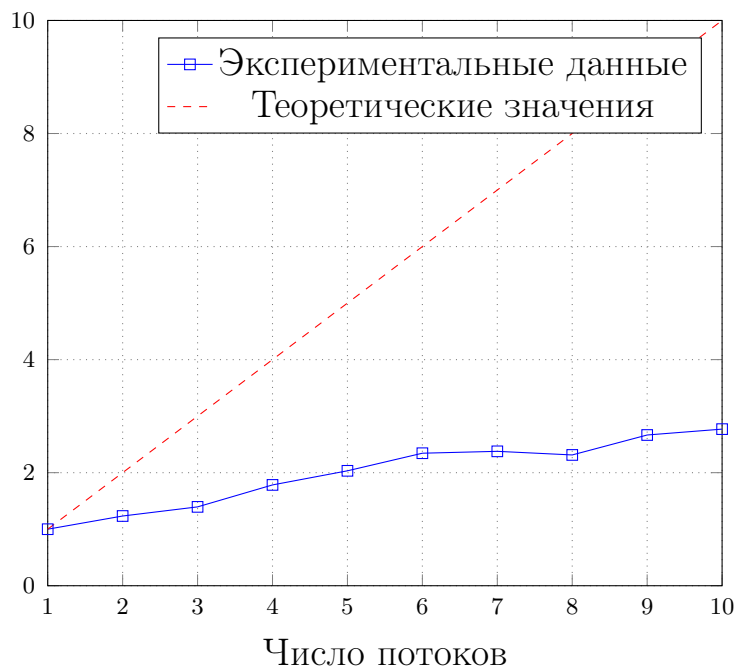
Эффективность / число потоков



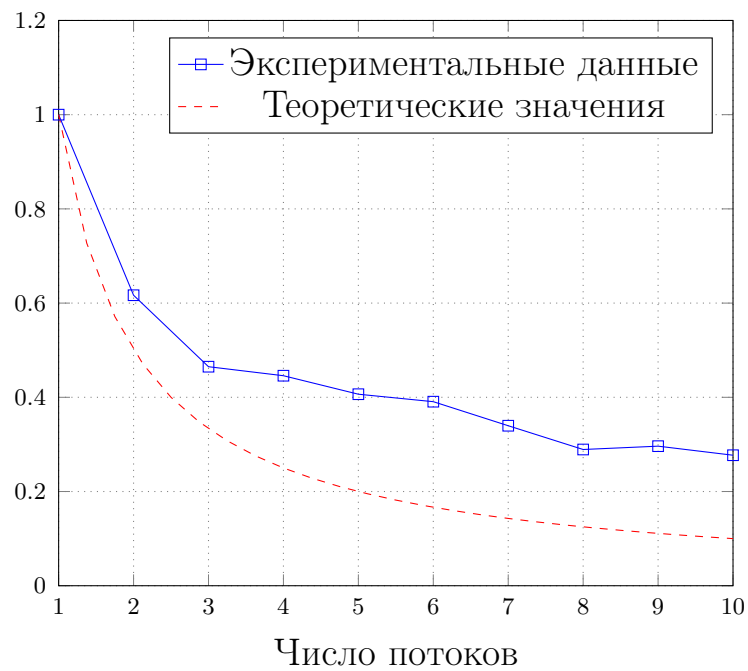
Случайный массив (Type 2)



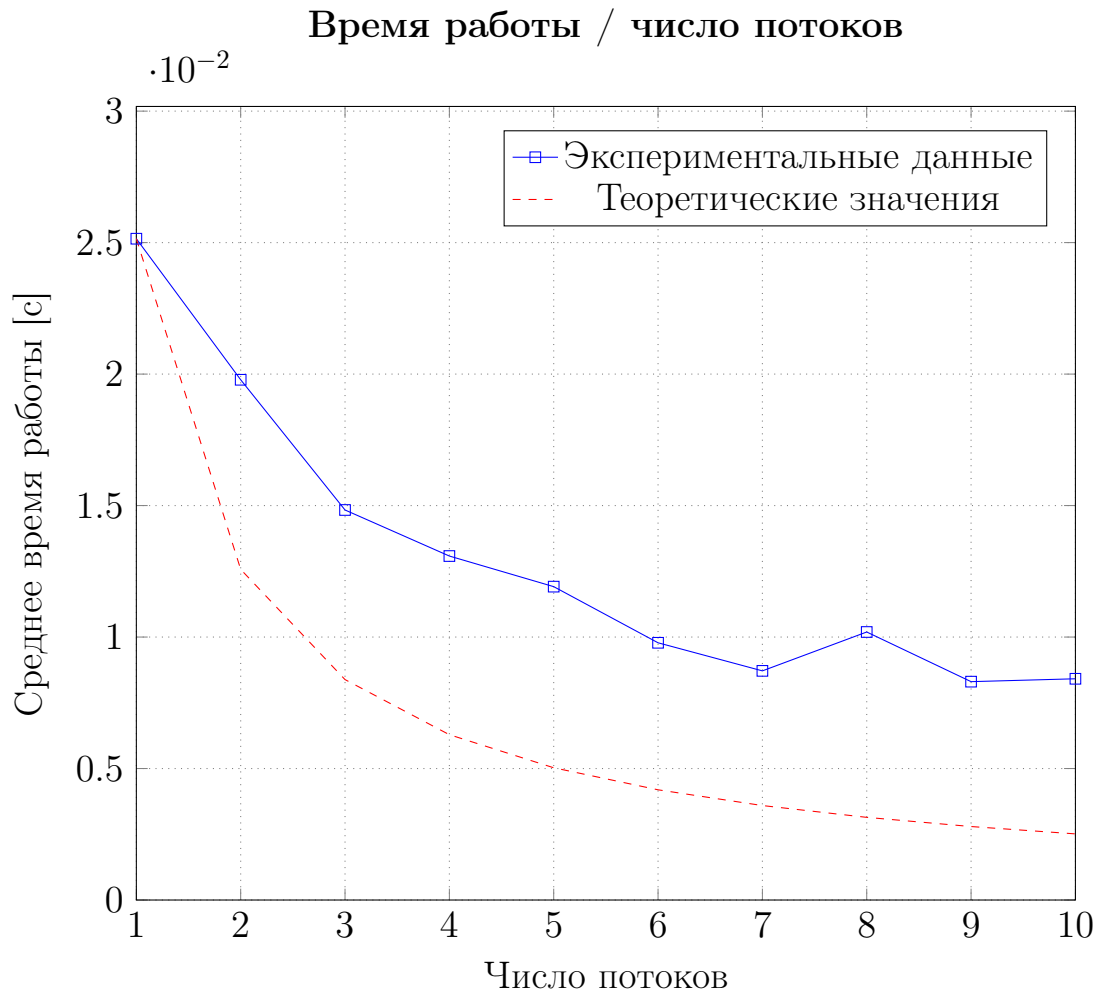
Ускорение / число потоков



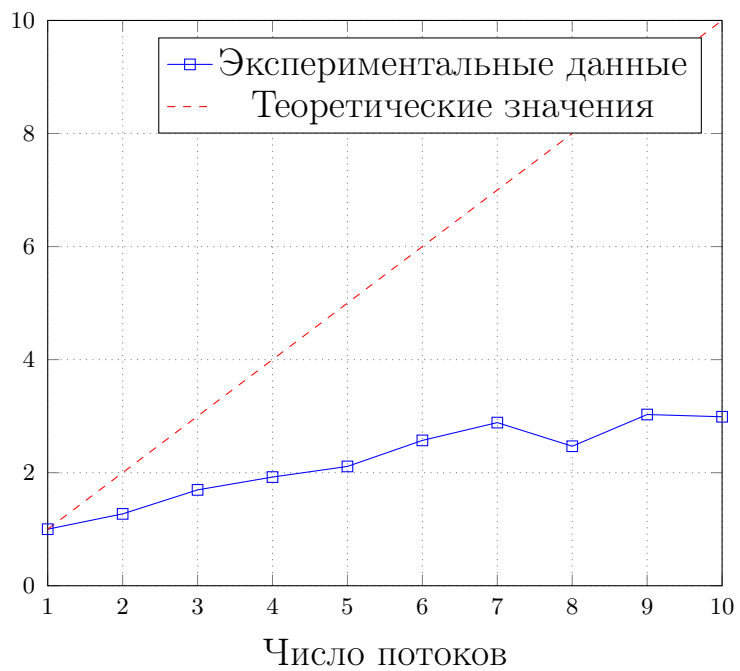
Эффективность / число потоков



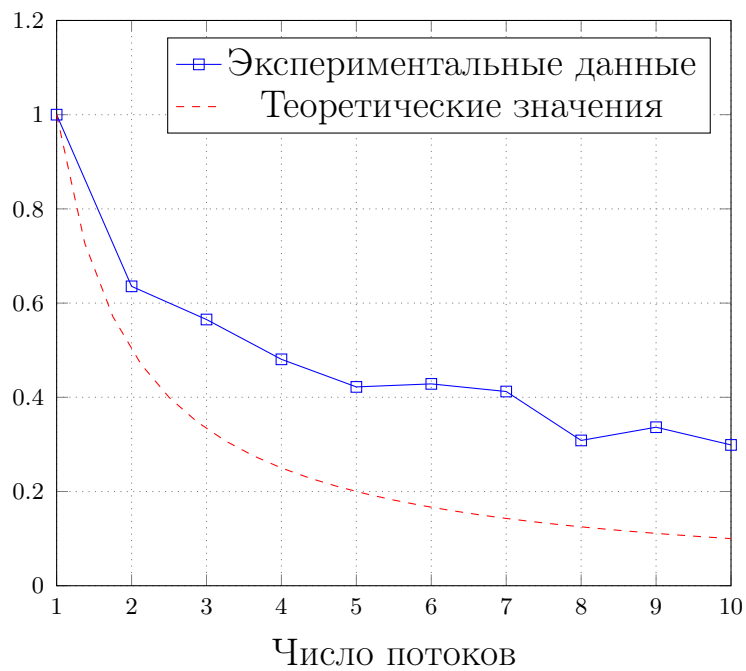
В массиве много одинаковых чисел (Type 3)



Ускорение / число потоков

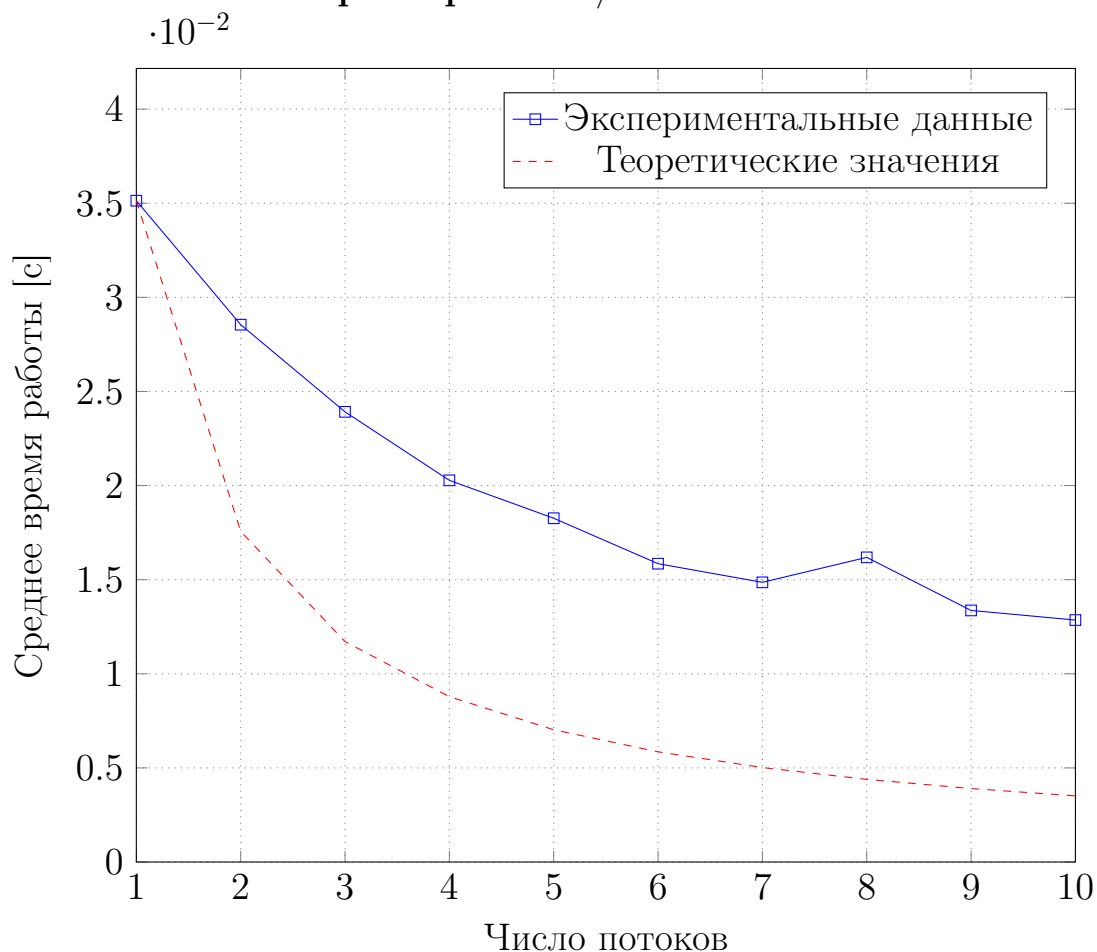


Эффективность / число потоков

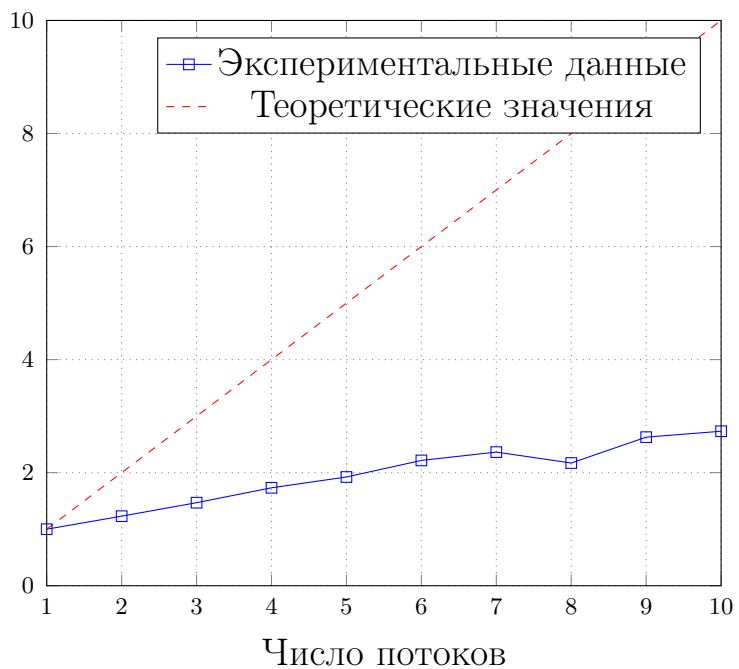


Первая половина массива упорядочена (Type 4)

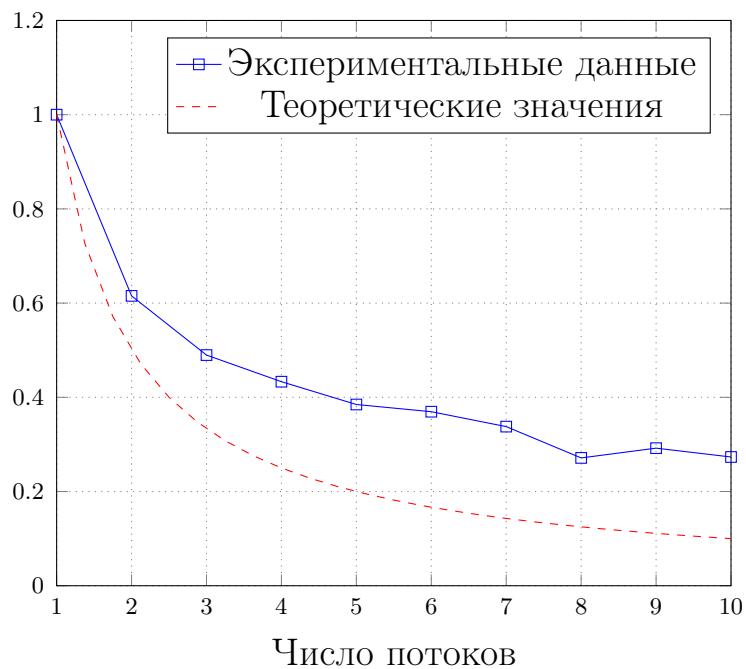
Время работы / число потоков



Ускорение / число потоков

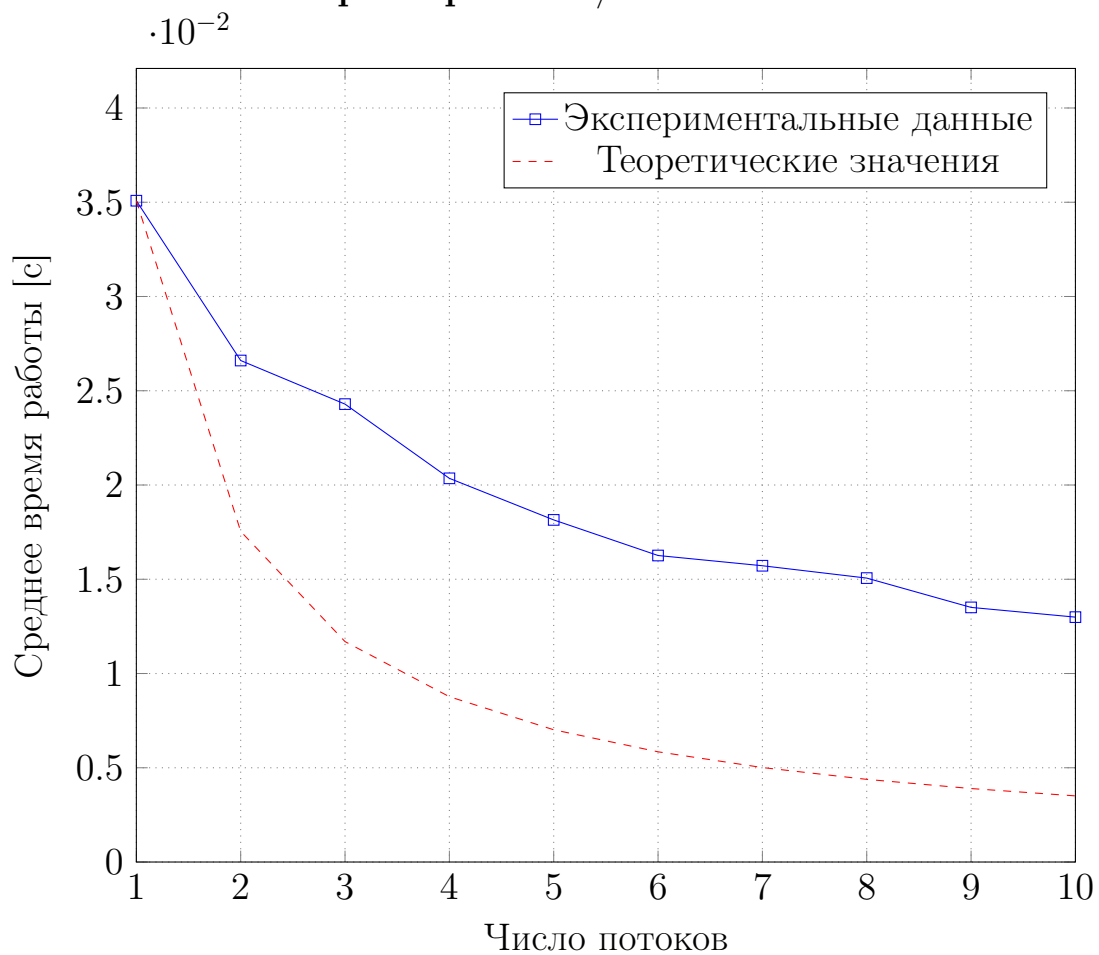


Эффективность / число потоков

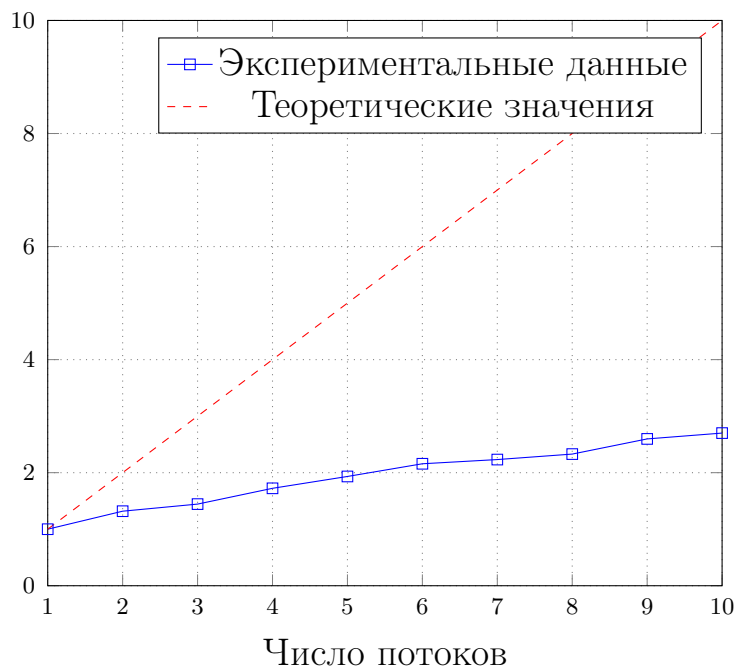


Первая половина массива обратно упорядочена (Туре 5)

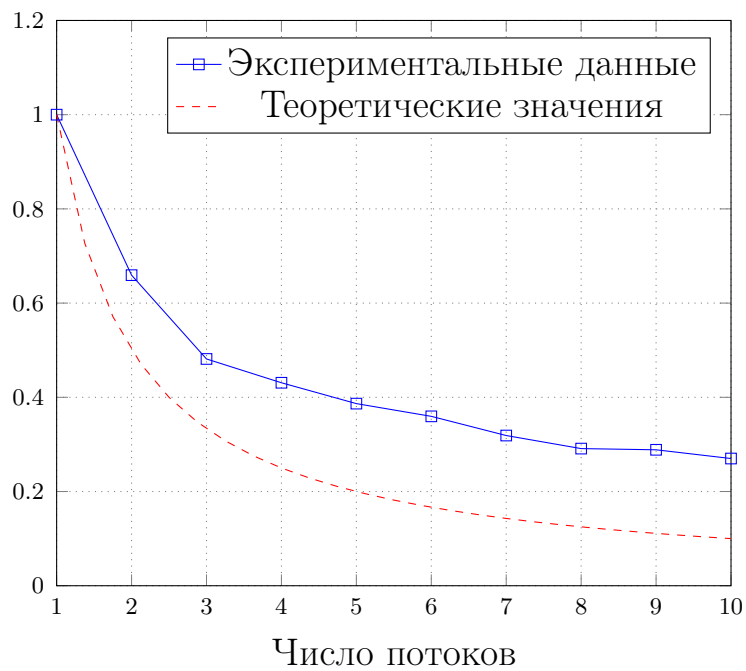
Время работы / число потоков



Ускорение / число потоков

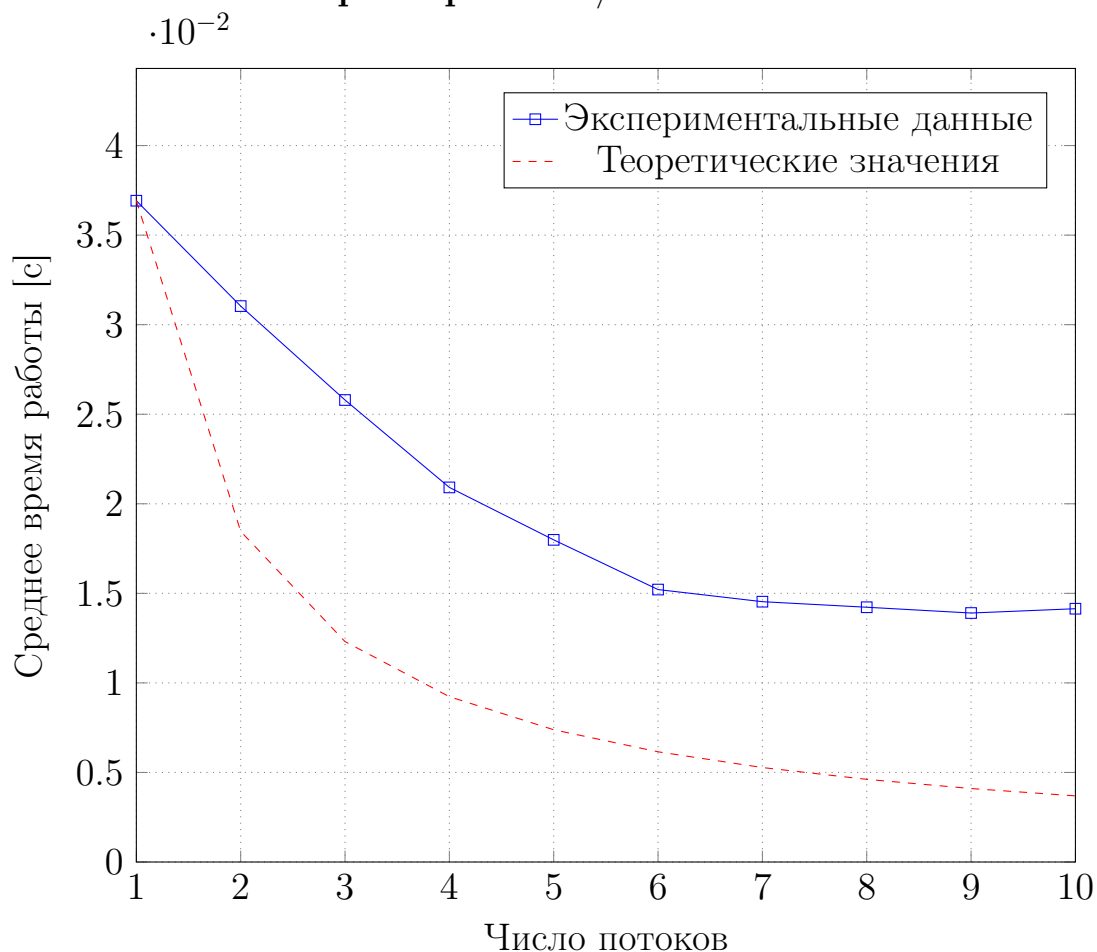


Эффективность / число потоков

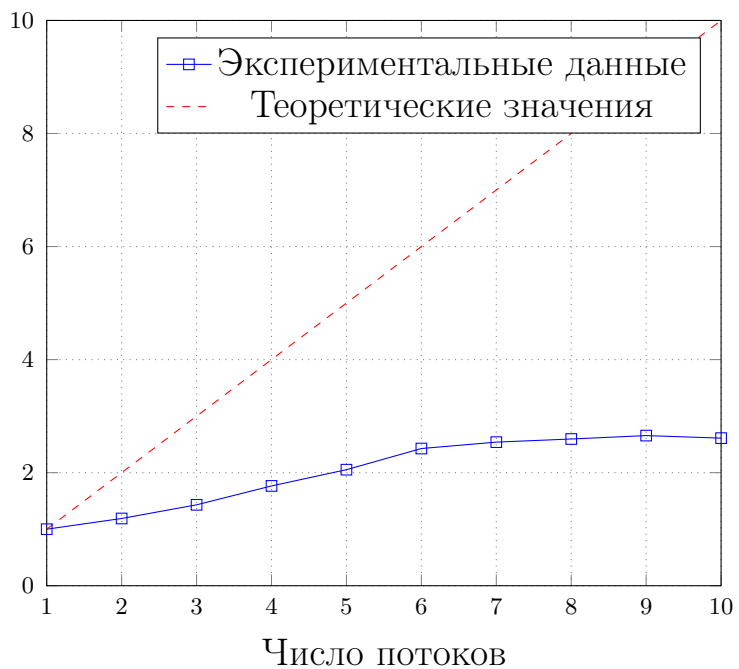


Первая четверть массива упорядочена (Туре 6)

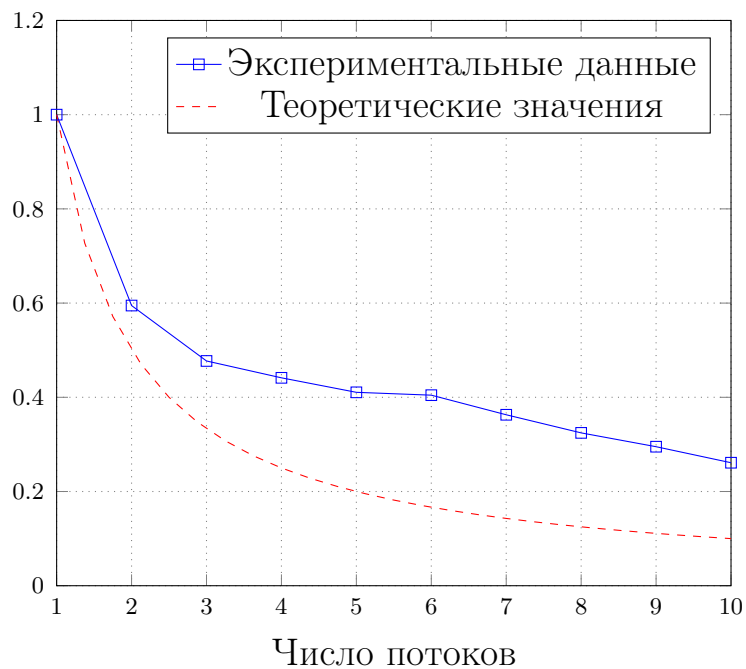
Время работы / число потоков



Ускорение / число потоков

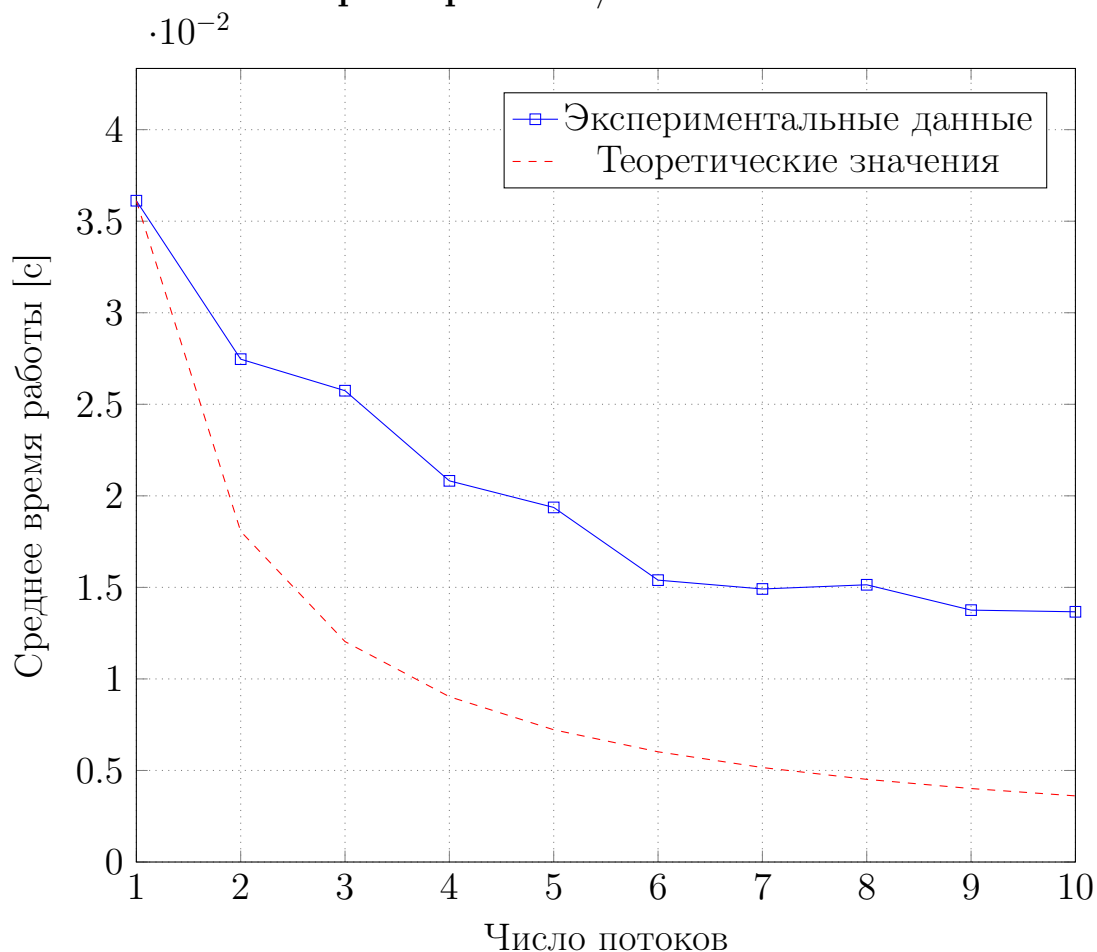


Эффективность / число потоков

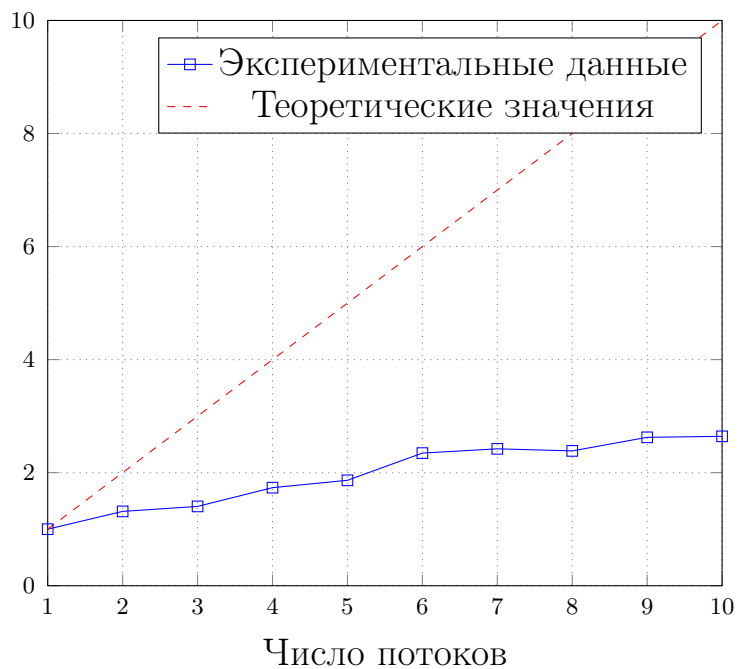


Первая четверть массива обратно упорядочена (Тип 7)

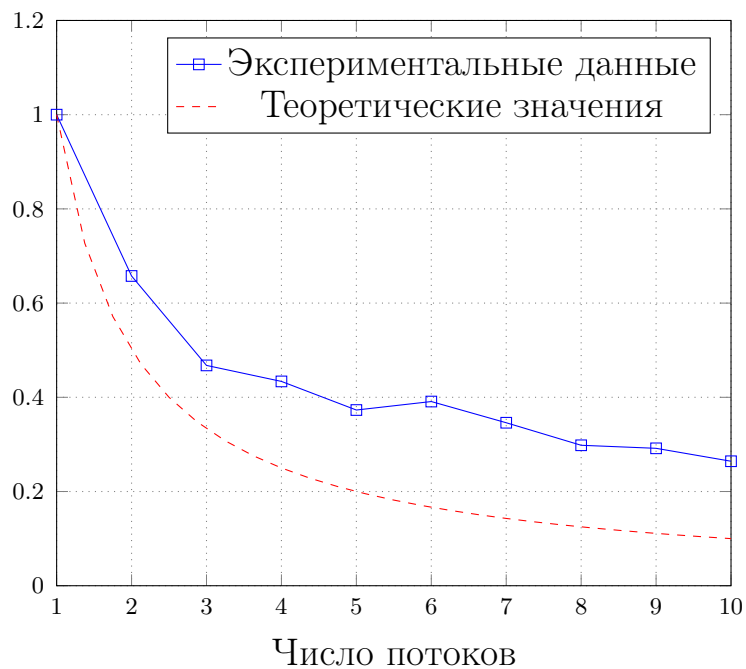
Время работы / число потоков



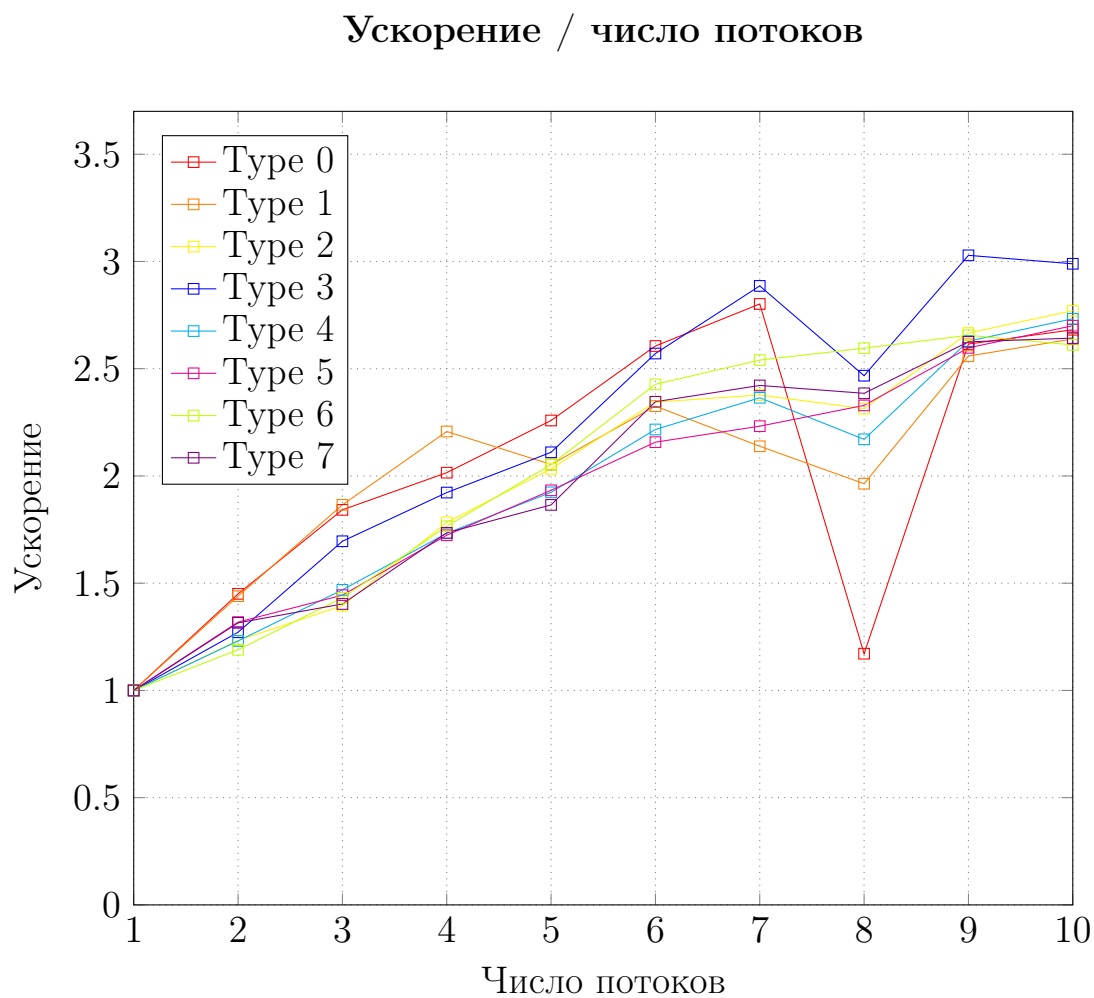
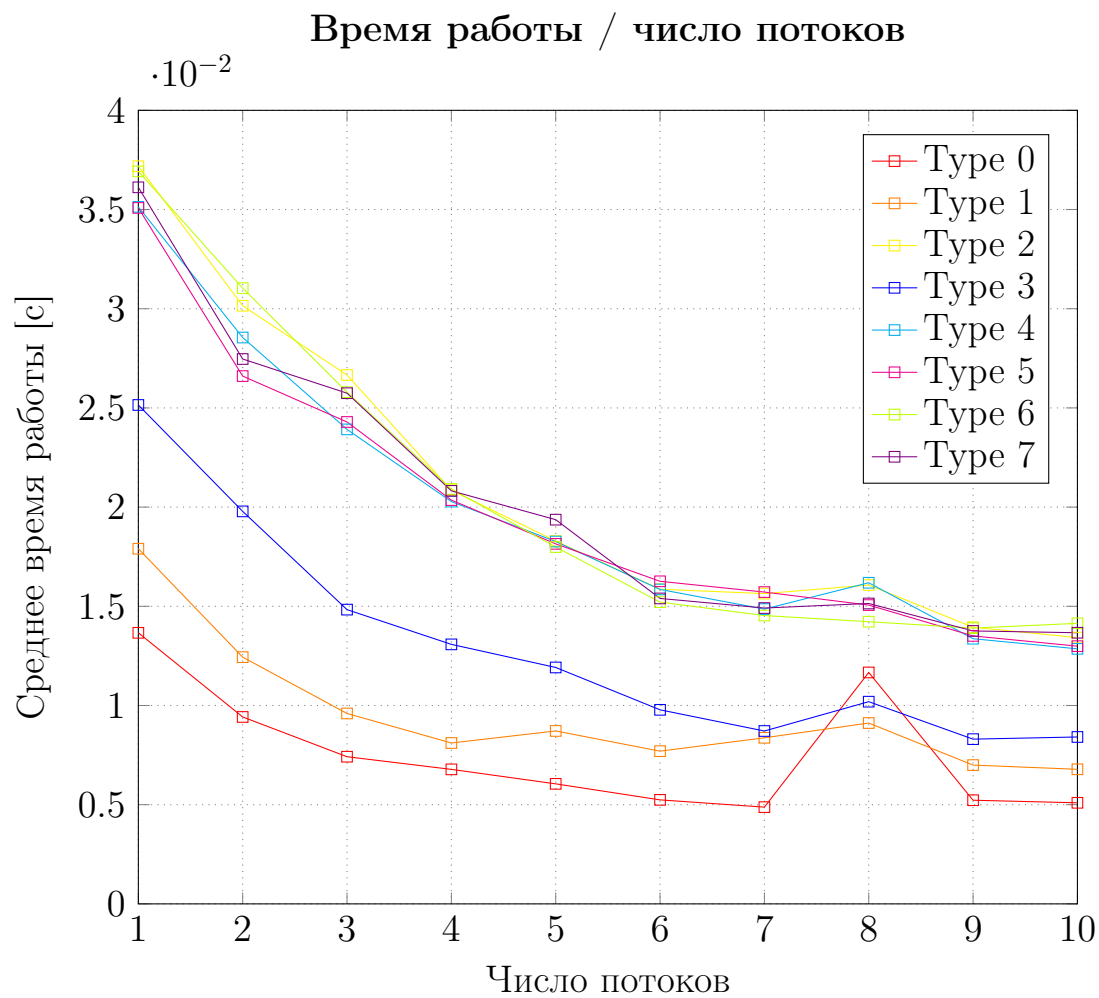
Ускорение / число потоков



Эффективность / число потоков



Сводные графики



5. Заключение

В ходе лабораторной работы была реализована сортировка Шелла на языке Си, выявлена область, допускающая распараллеливание - сортировка вставками непересекающихся подмассивов, реализован параллельный алгоритм сортировки Шелла.

Было измерено среднее время работы алгоритма для различного числа потоков и для различных типов массивов. По полученным данным были вычислены значения ускорения и эффективности. Построены соответствующие графики.

Анализируя результаты, можно сделать следующие выводы:

- Ускорение для любого числа потоков на всех типах массивов больше 1.
- Наилучший случай для линейного и параллельного алгоритма - упорядоченный массиве (время выполнения линейного алгоритма составляет 0.013664 с, наименьшее время выполнения параллельного алгоритма - 0.005095 с на 10 потоках).
- На обратно упорядоченном массиве время выполнения линейного алгоритма составляет 0.017902 с (на 31% медленнее линейной сортировки упорядоченного массива), хотя для сортировки вставками данный сценарий является наилучшим. Это достигается благодаря обмену элементов, находящихся на больших расстояниях на начальных этапах сортировки ("грубая сортировка"). Максимальное ускорение в этом случае составляет 2.638684 и достигается на 10 потоках.
- Характер зависимости ускорения и эффективности от числа потоков не меняется существенно на разных типах массивов (в отличие от предыдущей лабораторной работы).
- Наибольшее ускорение составляет 3.028589 и достигается в случае массива, содержащего много одинаковых чисел, на 9 потоках.

6. Приложение

Код программы, таблицы расположены на github

Запуск программы: `make all`