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 №6
«Коллективные операции в MPI»

Группа Б21-525

Студент Г.О. Шулындин

Преподаватель М.А. Куприяшин

Москва 2023

Оглавление

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 (В оперативной памяти есть проблемы. Требуется замена)
- RAM тип: DDR4
- Используемая среда разработки: Visual Studio Code
- Компилятор: gcc v11.4.0
- Поддерживаемая версия OpenMP: 201511
- Версия Open MPI: 4.0

2. 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).

3. OpenMPI

Построение параллельного алгоритма

Был реализован параллельный алгоритм сортировки Шелла. Сортируемый массив разбивается на подмассивы. Число подмассивов равно числу открытых параллельных процессов в группе. На каждом подмассиве выполняется сортировка Шелла. Используемый алгоритм сортировки Шелла:

```
void shell_sort_n_threads(int *array, int array_length) {
    int s, i;
    for (s = array_length / 2; s > 0; s /= 2) {
        for (i = 0; i < s; i++) {
            for (int j = i + s; j < array_length; 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;
            }
        }
    }
}
```

Затем полученные отсортированные массивы попарно соединяются, элементы располагаются в порядке возрастания. Операция соединения продолжается до тех пор, пока не будет получен массив исходной длины.

Временные характеристики

Следующая таблица содержит полученные в результате эксперимента данные: среднее время работы параллельного алгоритма сортировки Шелла, реализованного с помощью библиотеки OpenMPI, на различном числе параллельных процессов для массивов различных типов:

Thds num.	Type 0	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6	Type 7
1	0.013367	0.017408	0.037351	0.025551	0.035418	0.03516	0.038052	0.037882
2	0.007906	0.009338	0.019089	0.013349	0.019251	0.022072	0.019581	0.019696
3	0.006442	0.006854	0.014269	0.009922	0.01233	0.012701	0.014577	0.014278
4	0.005322	0.00591	0.011004	0.008097	0.011233	0.011047	0.01093	0.010788
5	0.006112	0.0064	0.01125	0.00816	0.009698	0.009794	0.010969	0.010833
6	0.005875	0.005527	0.010044	0.007482	0.008025	0.008146	0.009412	0.009468
7	0.006665	0.005288	0.009161	0.006379	0.008475	0.008443	0.008715	0.009341
8	0.007268	0.00597	0.011755	0.006166	0.010464	0.010974	0.0117	0.009583
9	0.029837	0.023875	0.023818	0.023964	0.022925	0.027771	0.028745	0.036847
10	0.055074	0.052242	0.027079	0.024712	0.026754	0.03075	0.03264	0.036782

4. OpenMP

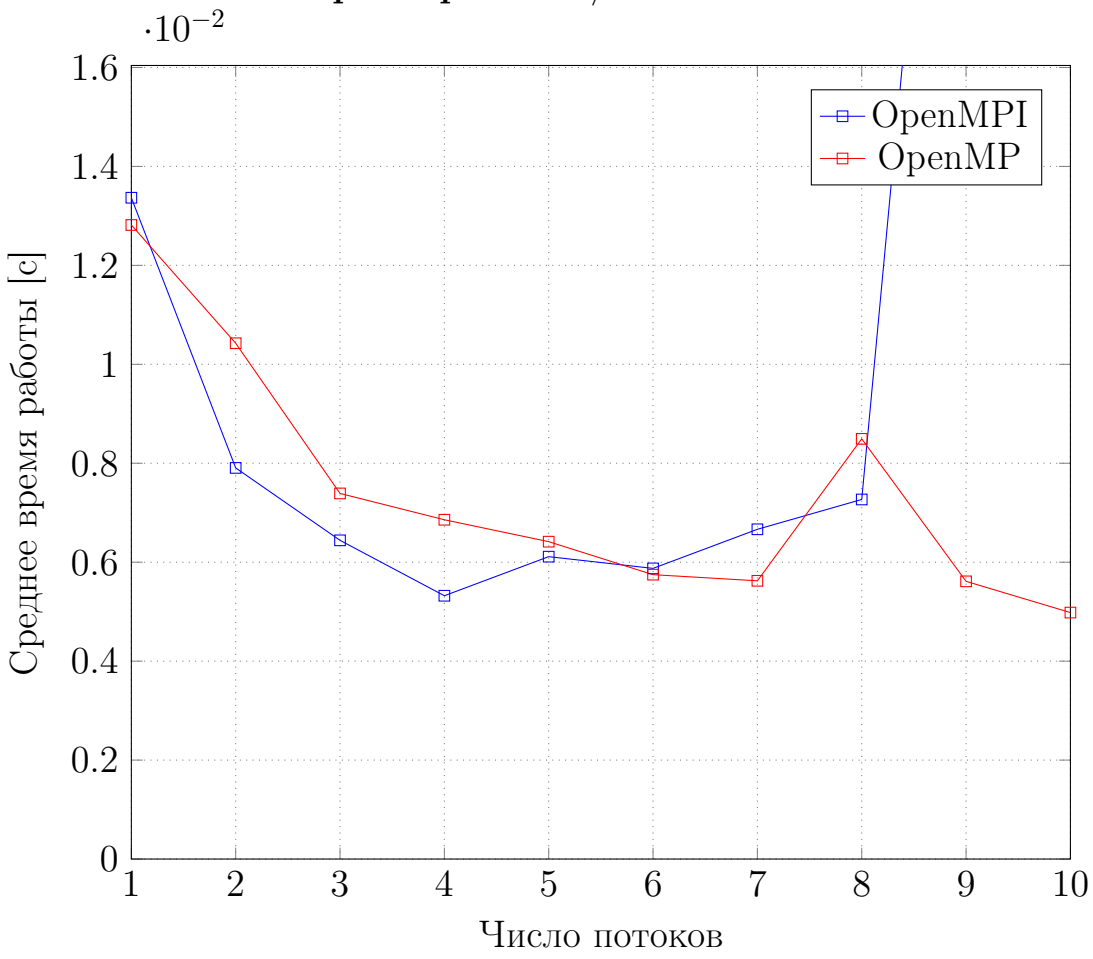
Для сравнения был взят параллельный алгоритм сортировки Шелла с использованием OpenMP из лабораторной работы №3. Среднее время работы на каждом числе потоков для каждого типа массивов:

Thds num.	Type 0	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6	Type 7
1	0.012816	0.017377	0.041023	0.025182	0.039332	0.036715	0.039094	0.040136
2	0.010426	0.013814	0.030507	0.015448	0.028354	0.028874	0.032253	0.033801
3	0.00739	0.011886	0.026754	0.015975	0.025564	0.025037	0.02548	0.026121
4	0.006858	0.00973	0.021898	0.012852	0.021128	0.021222	0.021858	0.021564
5	0.006415	0.008821	0.019658	0.011409	0.018528	0.018962	0.019256	0.019006
6	0.005748	0.009031	0.017366	0.010675	0.017187	0.017386	0.017323	0.016338
7	0.005624	0.009081	0.014185	0.00883	0.014969	0.01431	0.01549	0.016377
8	0.008493	0.009313	0.015513	0.009732	0.016736	0.014688	0.014627	0.013386
9	0.005613	0.007292	0.014404	0.008432	0.013608	0.013808	0.013979	0.013737
10	0.004982	0.006824	0.013594	0.00774	0.013043	0.013178	0.013708	0.013366

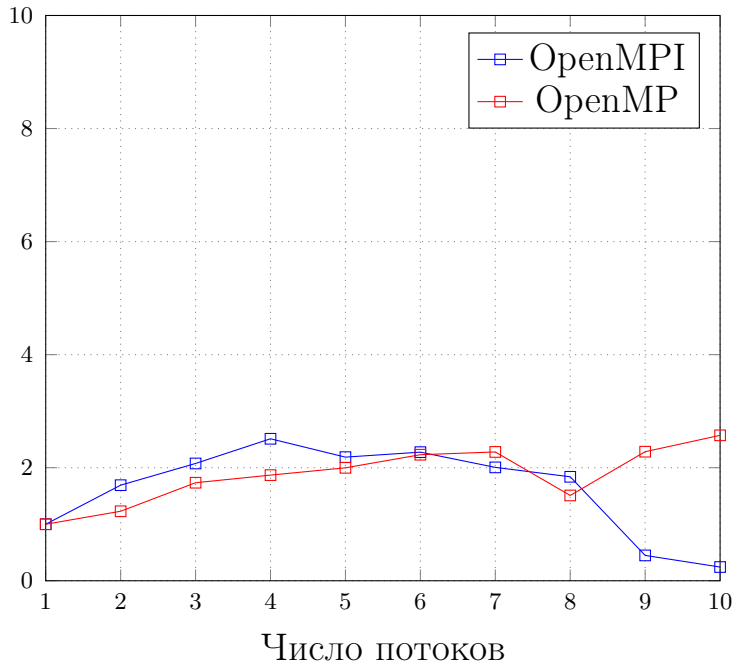
5. Сравнительные графики

Массив упорядочен (Type 0)

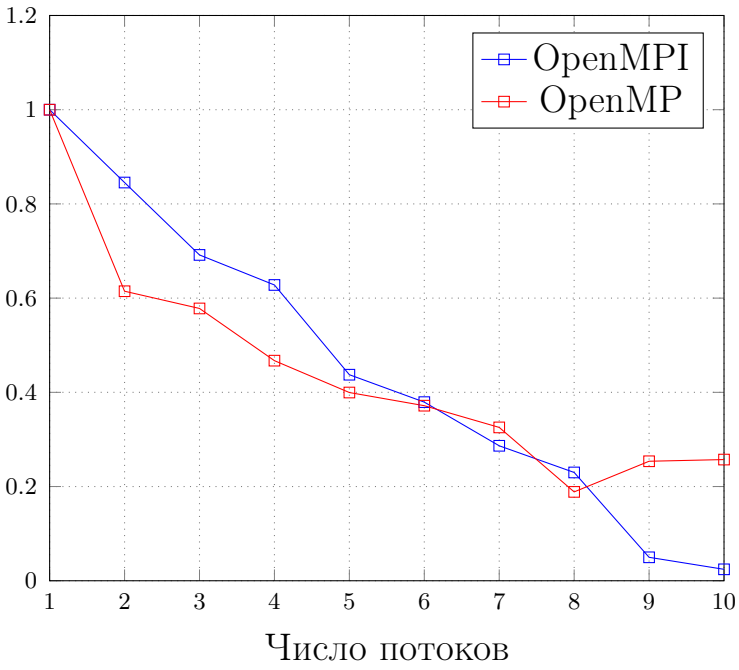
Время работы / число потоков



Ускорение / число потоков

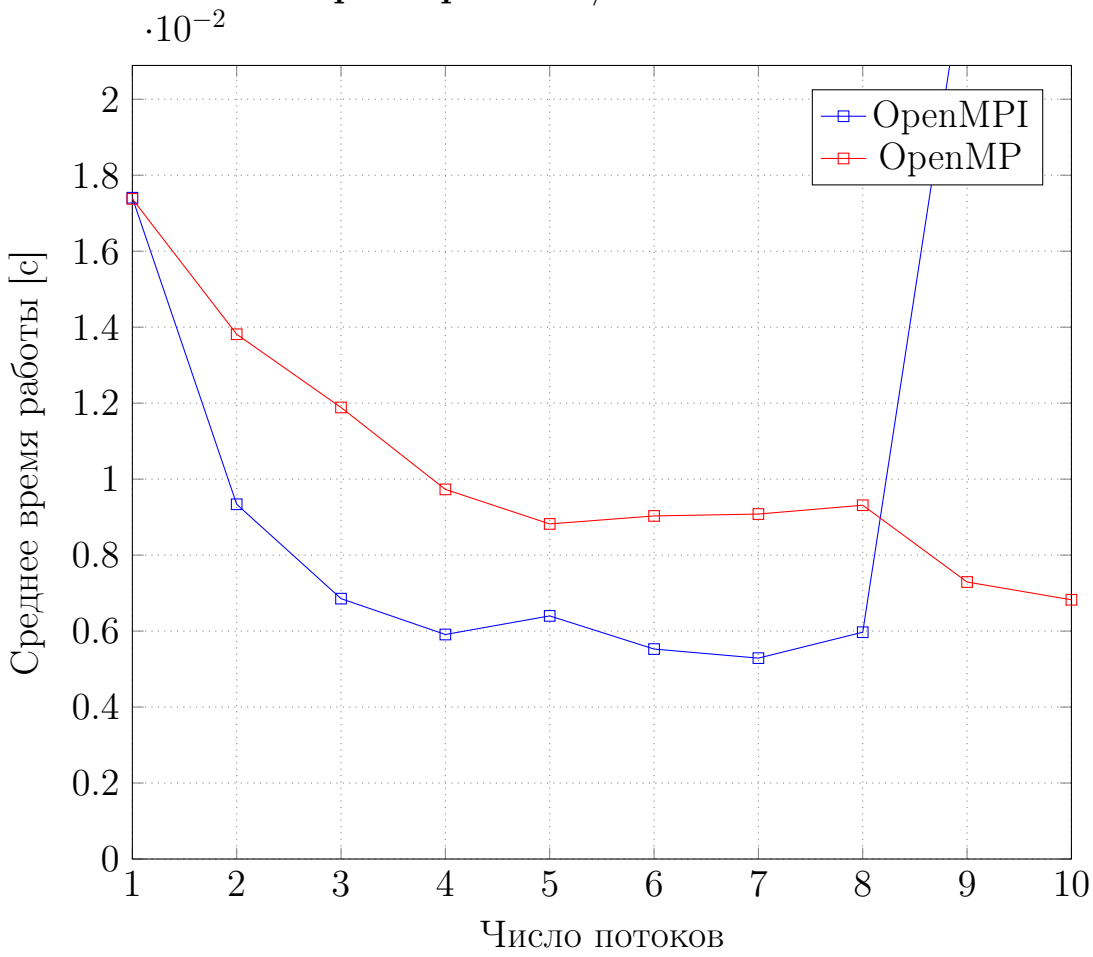


Эффективность / число потоков

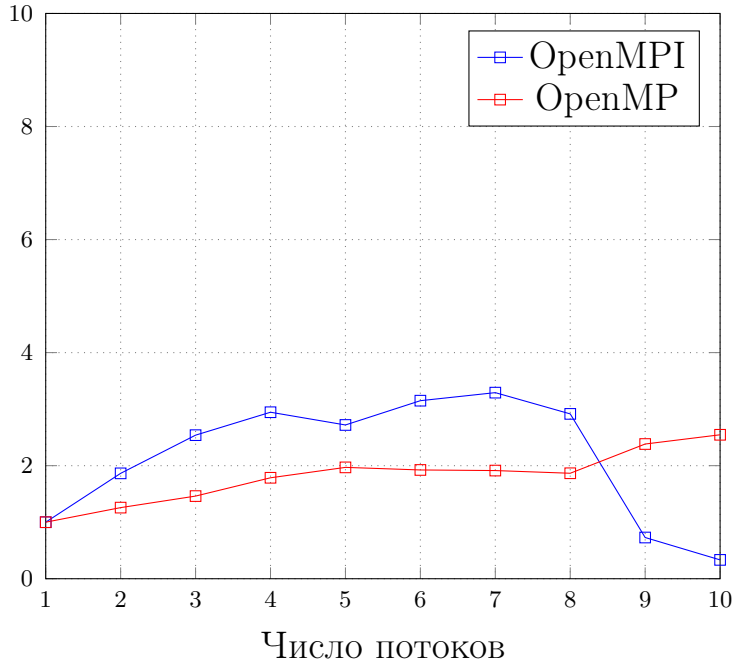


Массив обратно упорядочен (Type 1)

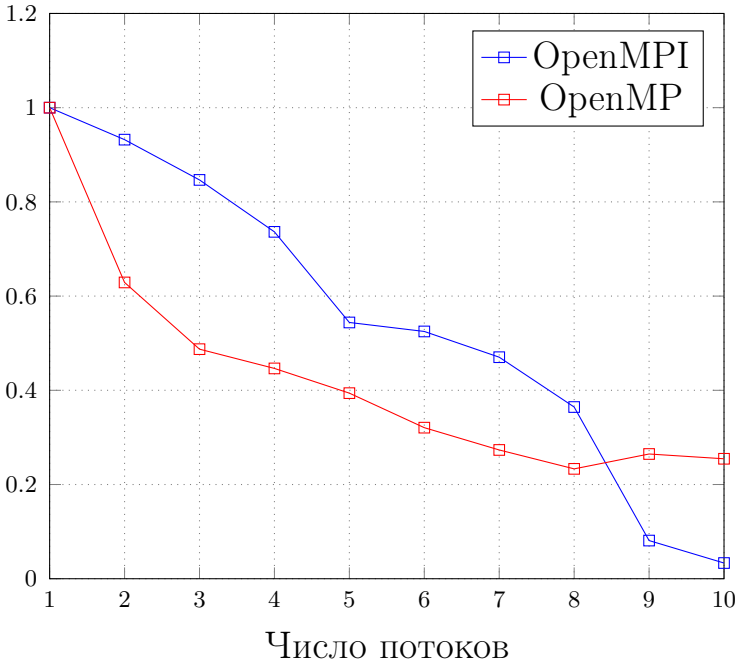
Время работы / число потоков



Ускорение / число потоков

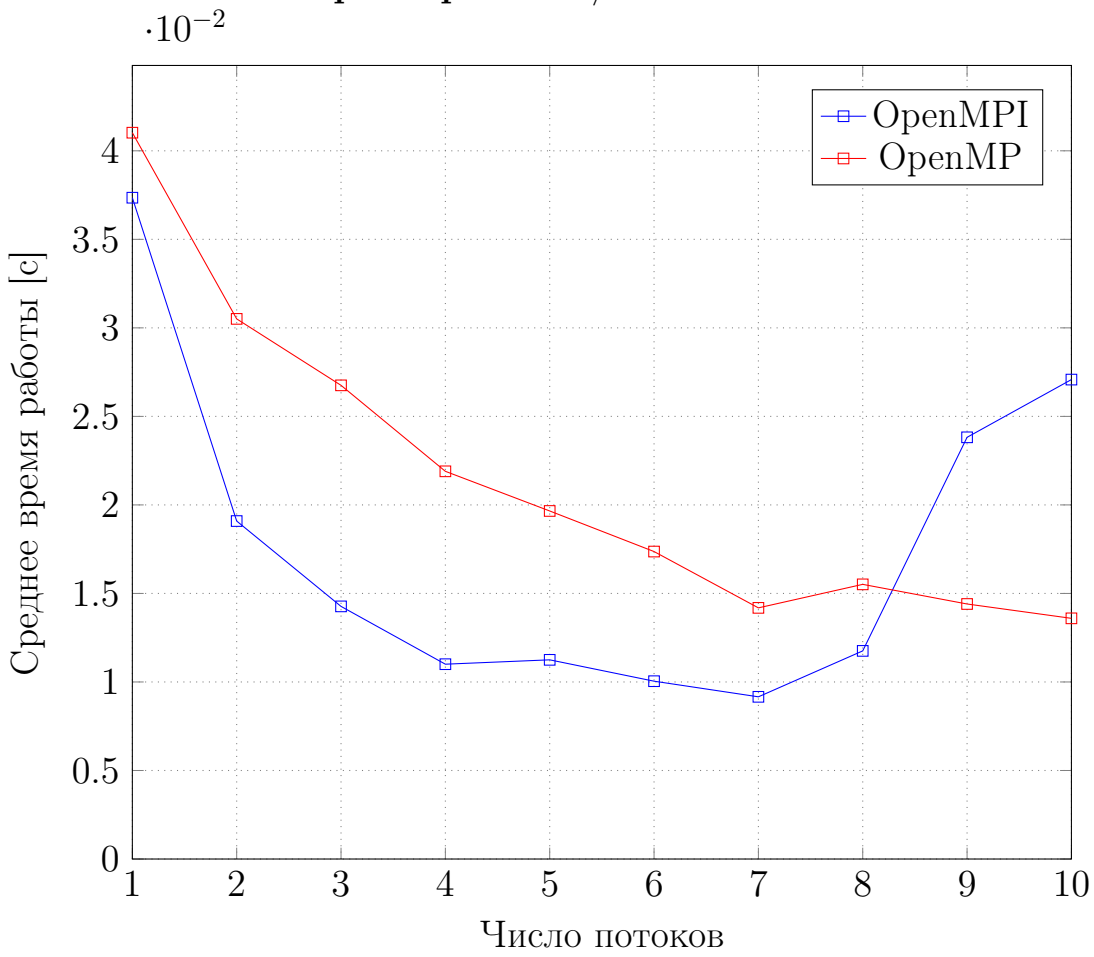


Эффективность / число потоков

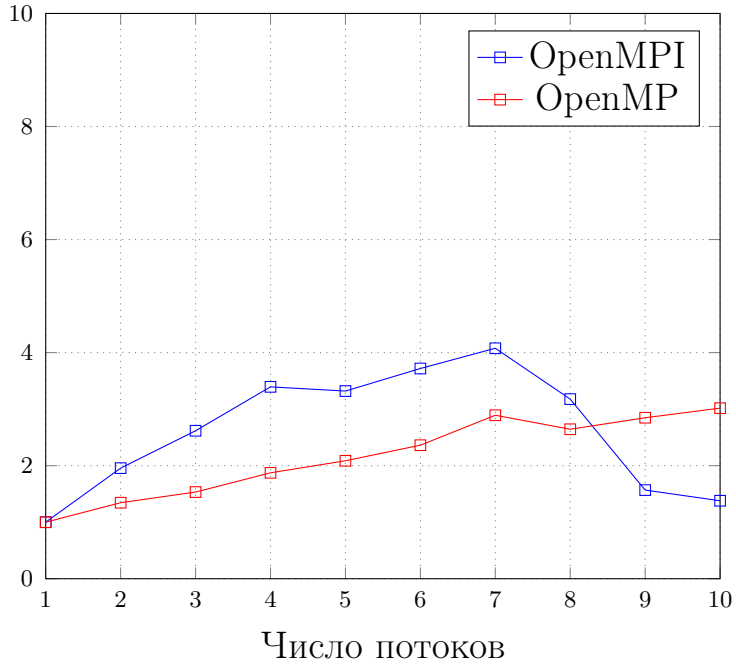


Случайный массив (Type 2)

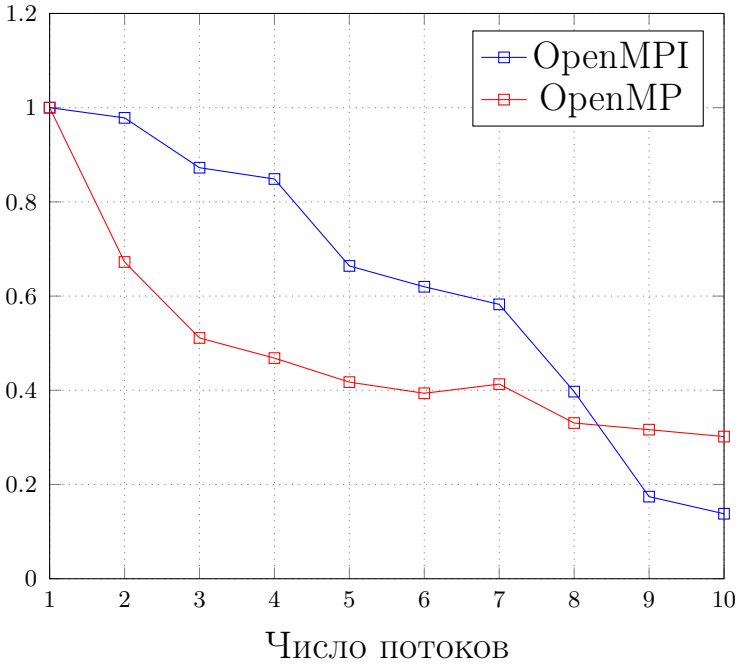
Время работы / число потоков



Ускорение / число потоков

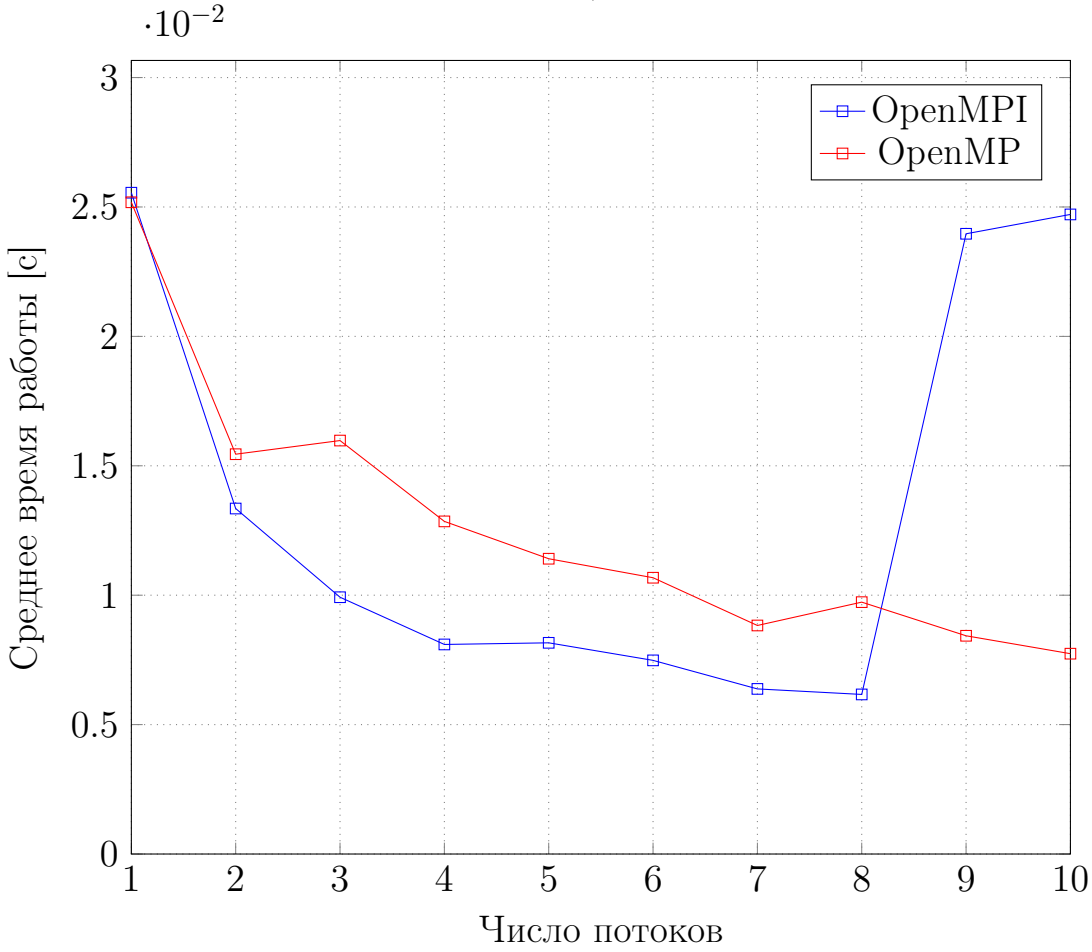


Эффективность / число потоков

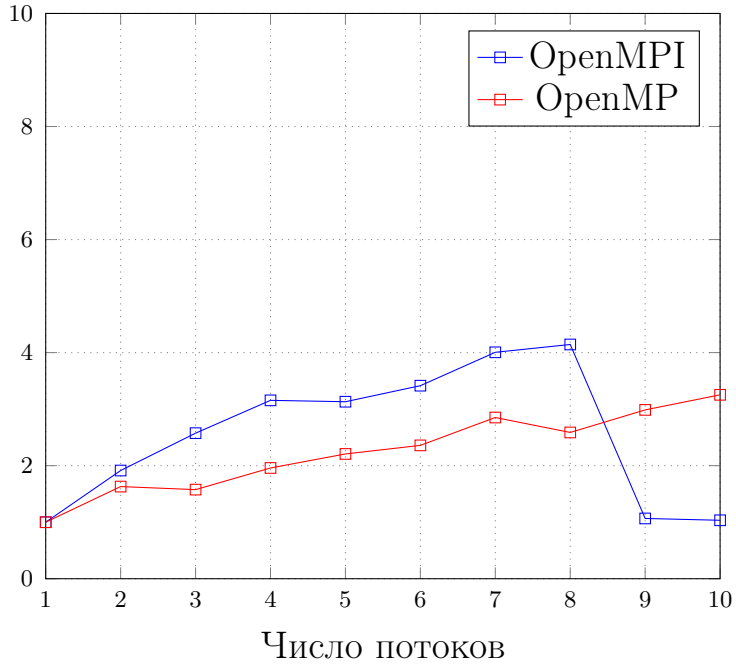


В массиве много одинаковых чисел (Type 3)

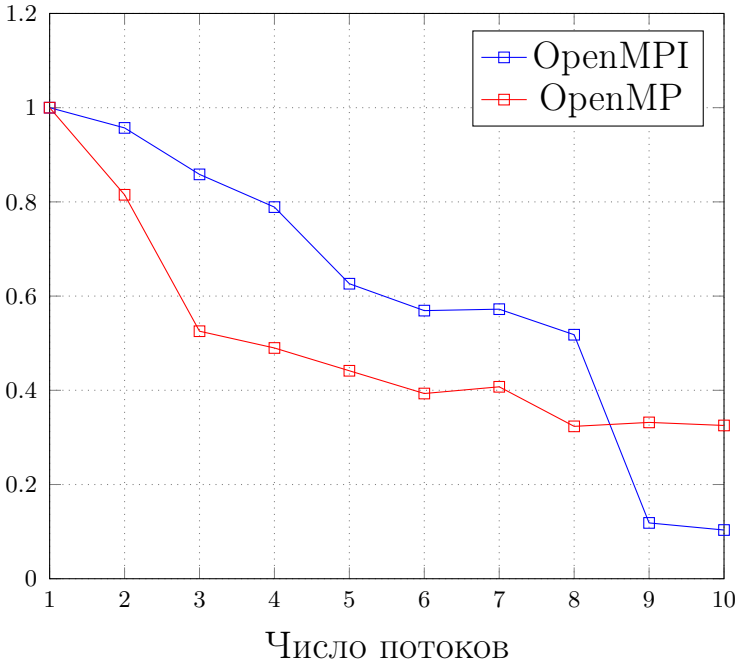
Время работы / число потоков



Ускорение / число потоков

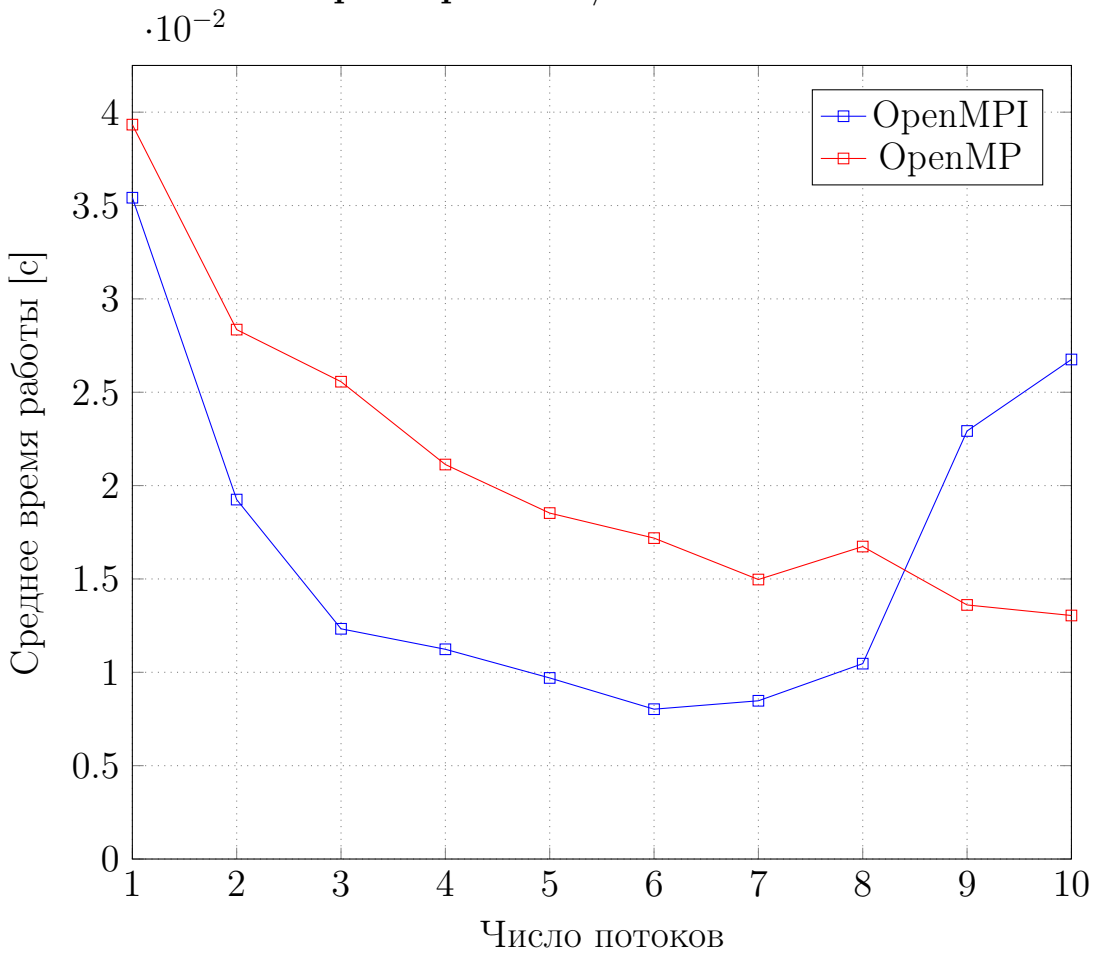


Эффективность / число потоков

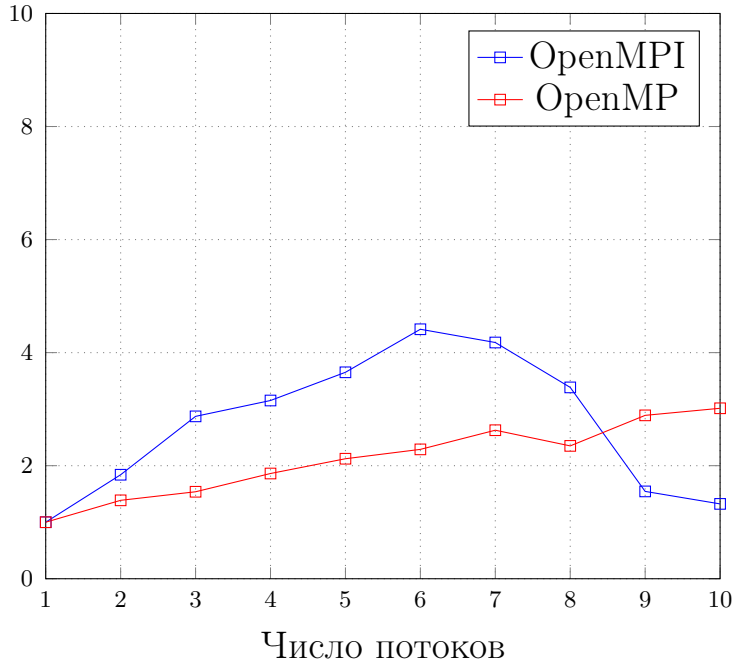


Первая половина массива упорядочена (Type 4)

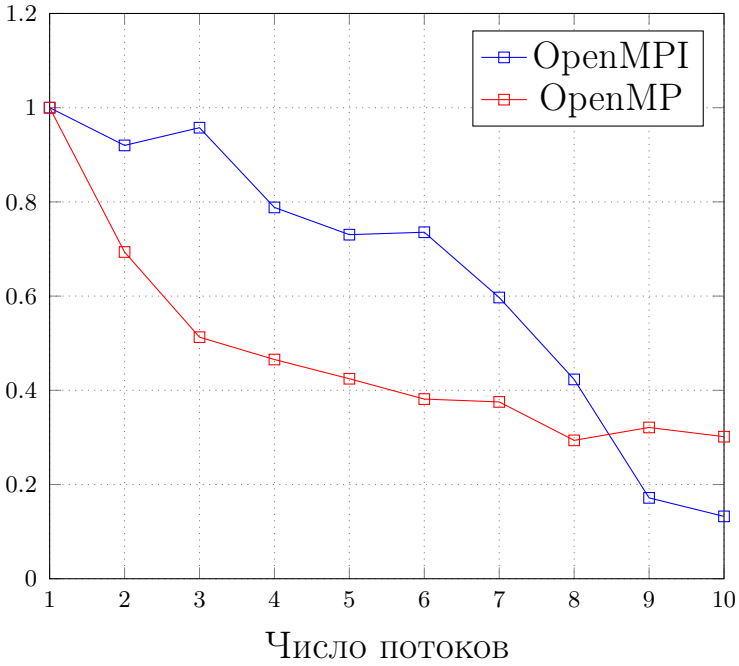
Время работы / число потоков



Ускорение / число потоков

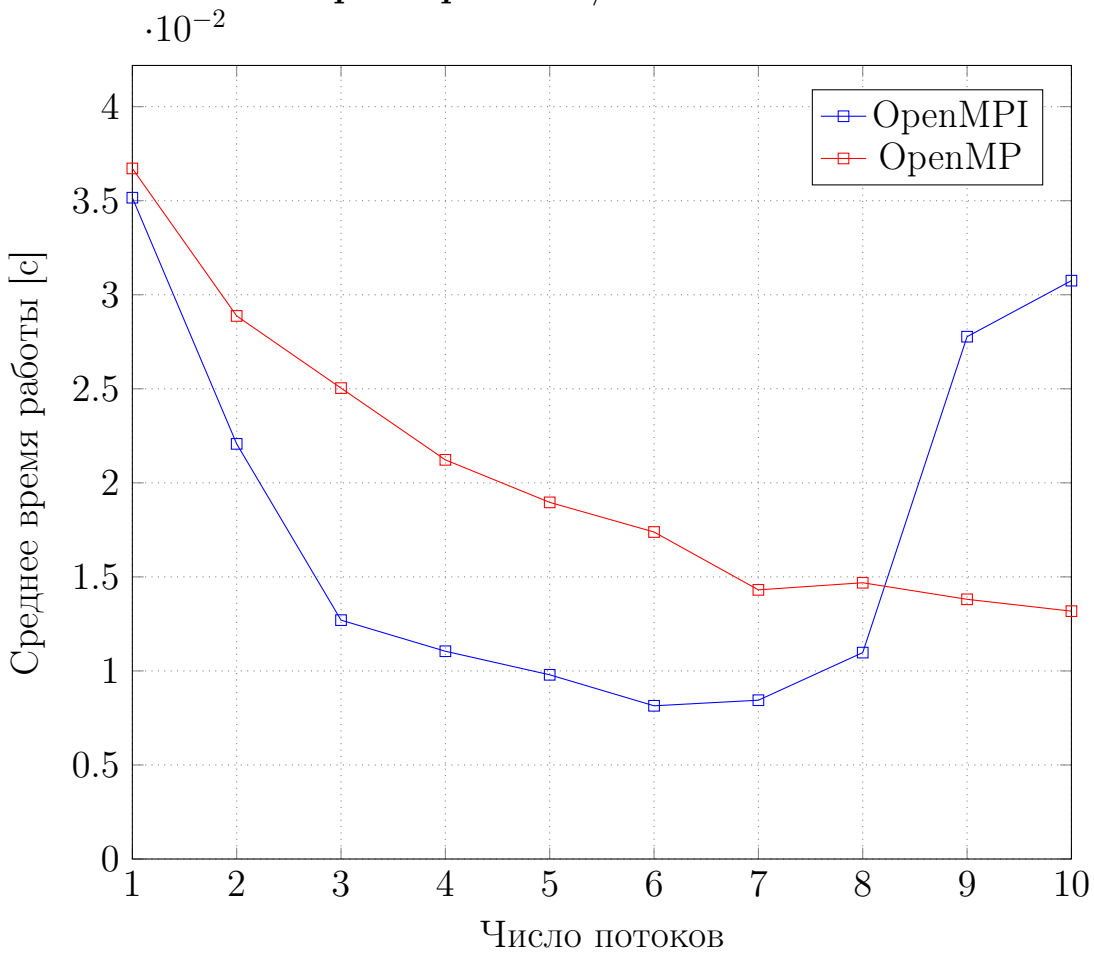


Эффективность / число потоков

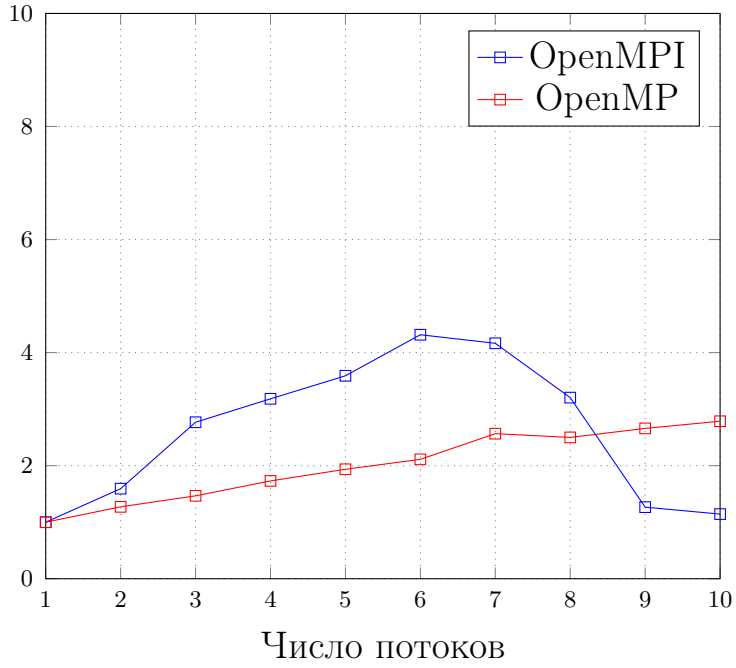


Первая половина массива обратно упорядочена (Type 5)

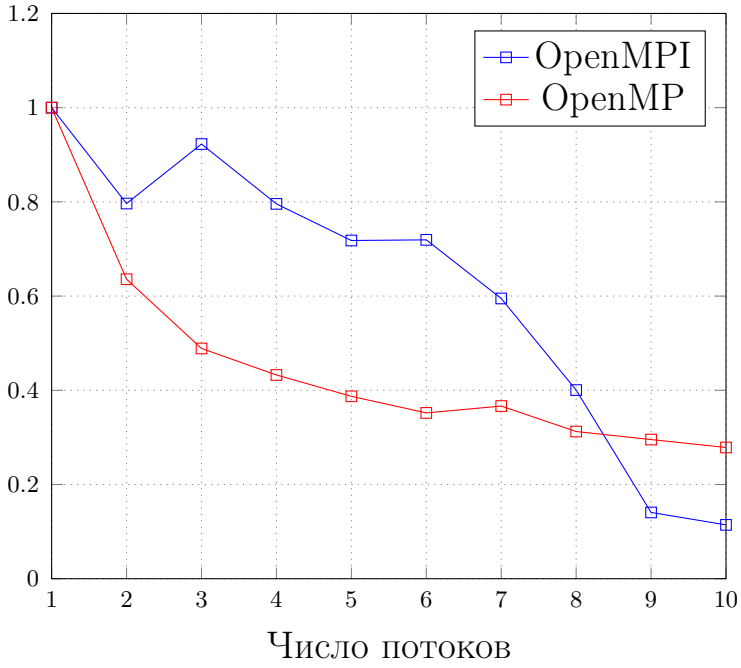
Время работы / число потоков



Ускорение / число потоков

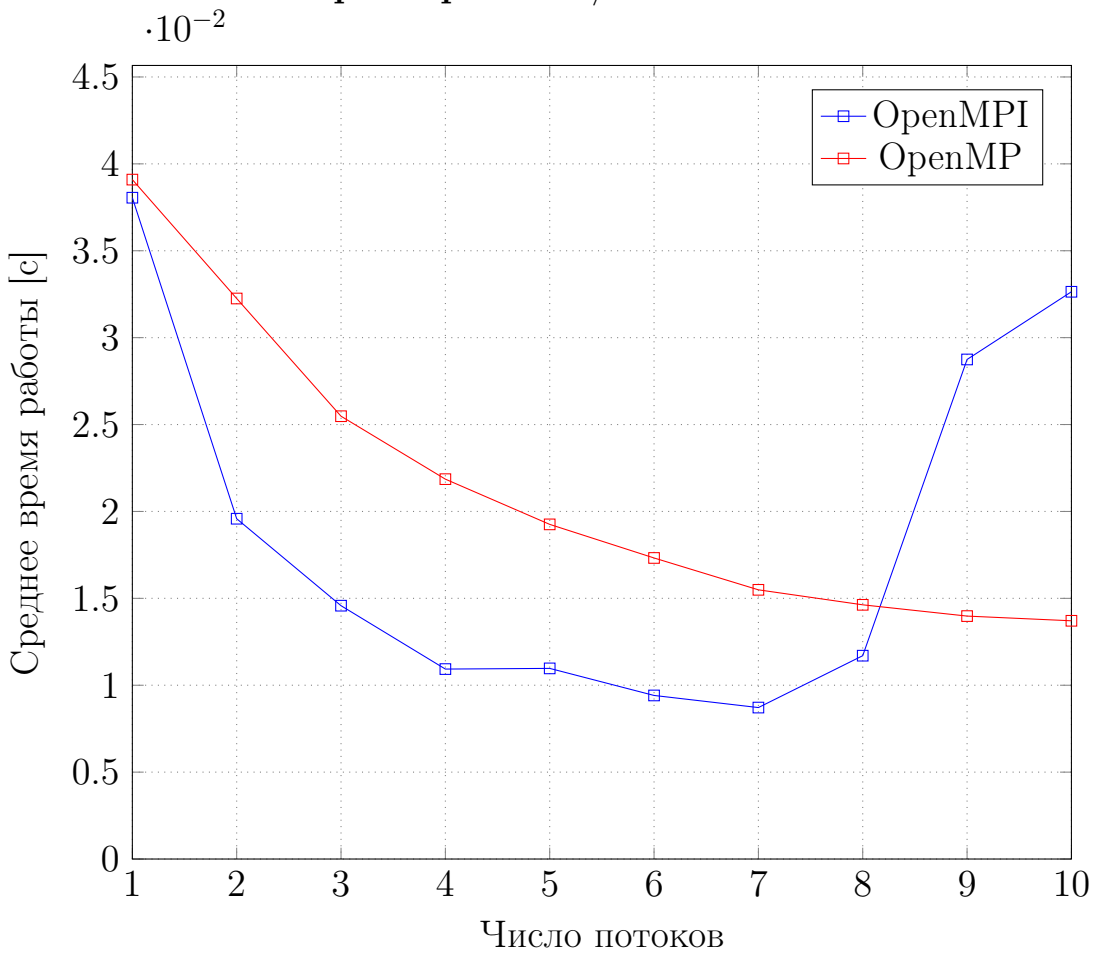


Эффективность / число потоков

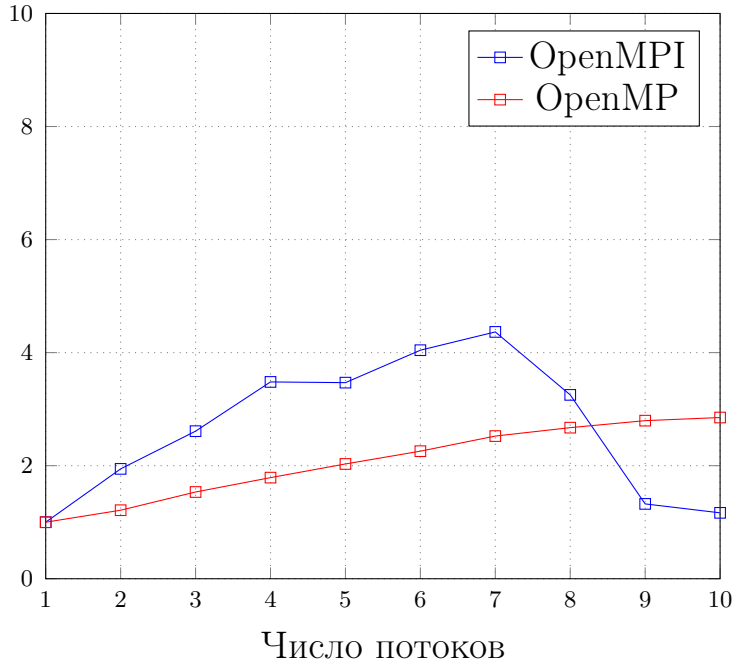


Первая четверть массива упорядочена (Type 6)

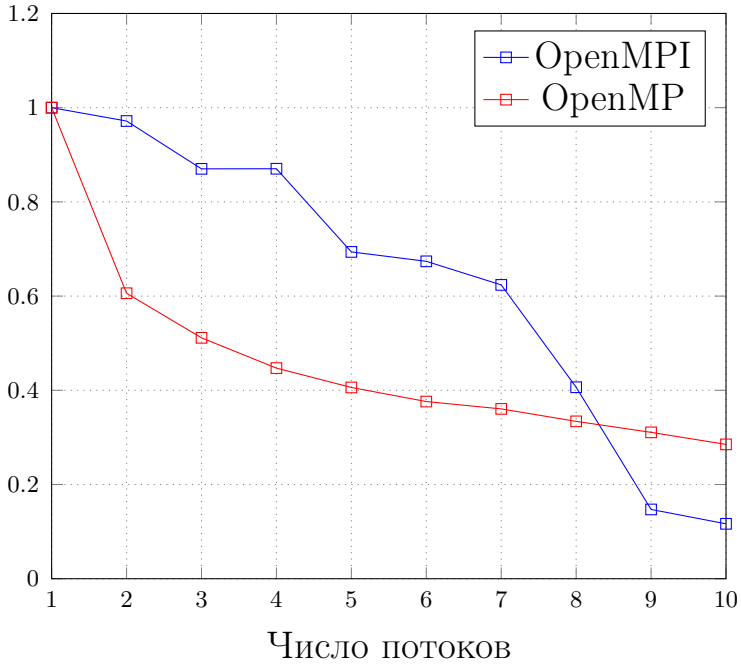
Время работы / число потоков



Ускорение / число потоков

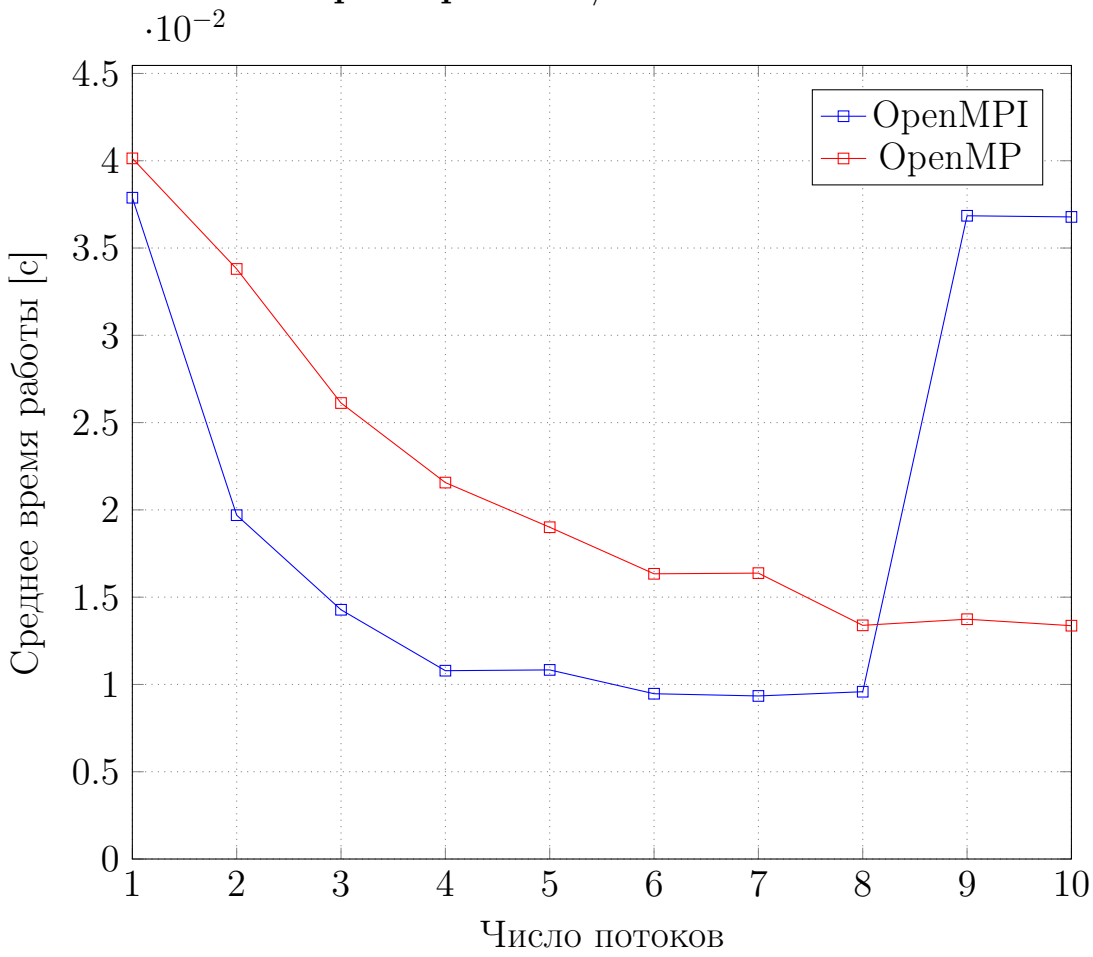


Эффективность / число потоков

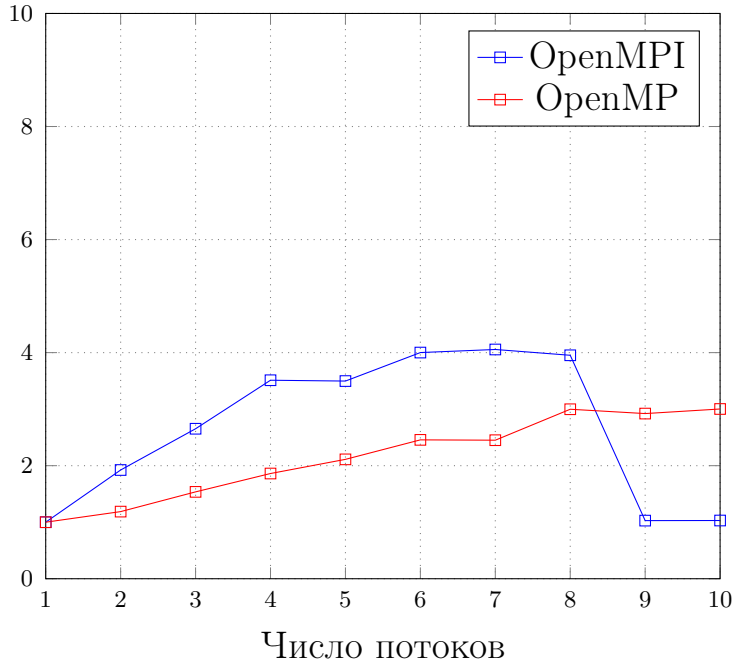


Первая четверть массива обратно упорядочена (Type 7)

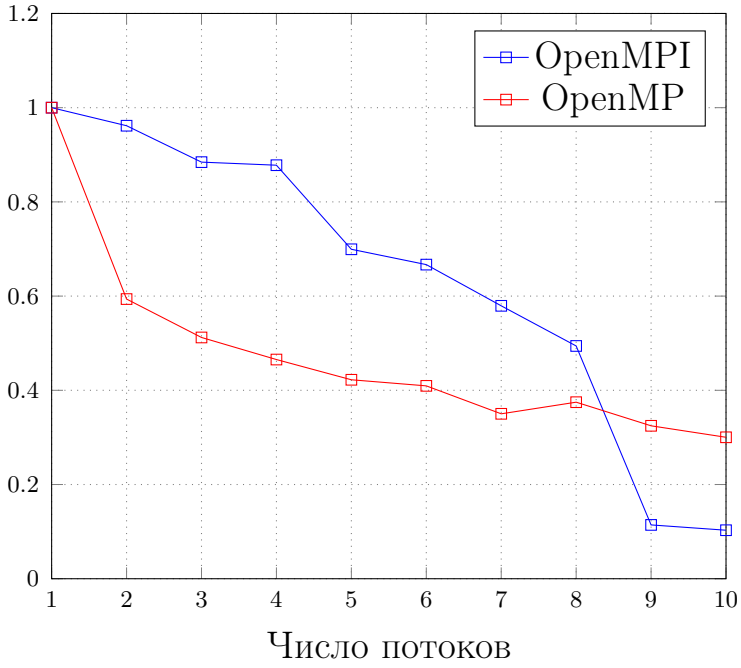
Время работы / число потоков



Ускорение / число потоков



Эффективность / число потоков



6. Заключение

В ходе лабораторной работы был реализован параллельный алгоритм поиска максимального элемента в массиве с использованием библиотеки Open MPI. Также был реализован тот же алгоритм, но с использованием технологии OpneMP. Было измерено среднее время работы алгоритмов на разном числе потоков. Были вычислены значения ускорения и эффективности, построены соответствующие графики.
На основе этих данных можно сделать следующие выводы:

- Для числа потоков от 1 до 7 время работы алгоритма с использованием Open MPI примерно равно времени работы алгоритма с использованием OpenMP. Максимальное в этом промежутке числа потоков ускорение при использовании Open MPI составляет 5.17 на 6 потоках; при использовании OpenMP 4.16 на 7 потоках.
- Для числа потоков больше 7 время работы при использовании OpenMP остается примерно одинаковым, а при использовании Open MPI начинает расти.

7. Приложение

Код программы расположен на github
Запуск программы: `make all`