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 №1
«Введение в параллельные вычисления. Технология OpenMP»

Группа Б21-525

Студент Г.О. Шулындин

Преподаватель М.А. Куприяшин

Москва 2023

Оглавление

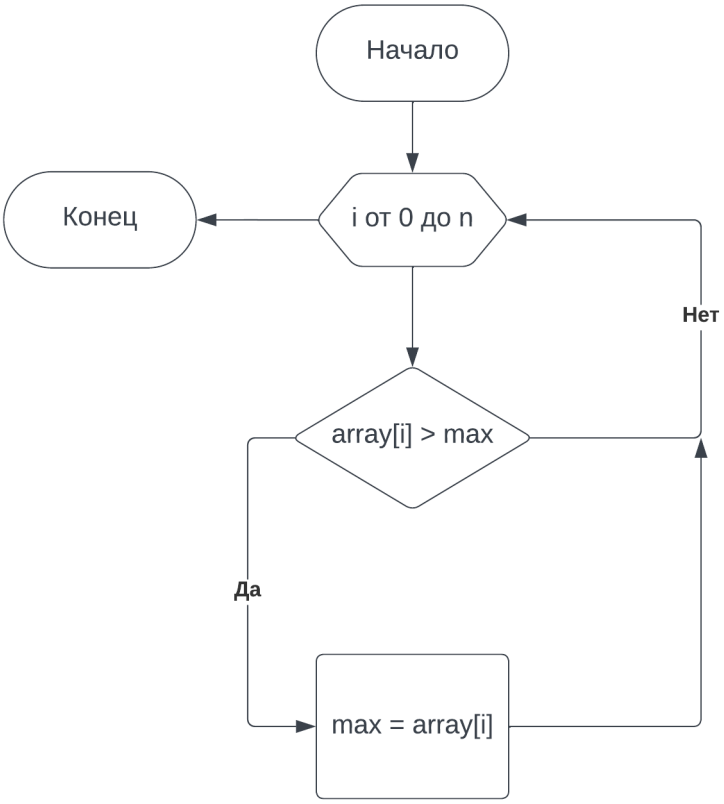
1. Описание рабочей среды

- Модель процессора: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx
- Число ядер: 8
- Архитектура: x86-64
- ОС: Linux, дистрибутив Ubuntu v20.04
- RAM объем: 2x4096 MB
- RAM тип: DDR4
- Используемая среда разработки: Visual Studio Code
- Компилятор: gcc v9.4.0
- Поддерживаемая версия OpenMP: 201511

2. Анализ приведенного алгоритма

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

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;
- `reduction()` - определяет одну или несколько переменных, являющихся приватными для каждого потока, которые подвергнутся операции редукции при выходе из параллельной области;
- `shared()` - объявляет, что одна или несколько переменных должны быть общими между всеми потоками;
- `default()` - задаёт стандартное поведение при встрече неинициализированной внутри потока переменной.

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

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

```
{
#pragma omp for
for(int i=0; i<count; i++) {
    if(array[i] > max) { max = array[i]; };
}
}
```

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

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

Описание работы алгоритма

Сначала директивой `parallel` объявляется блок кода, который будет исполняться параллельно. Далее благодаря директиве `for` внутри происходит распараллеливание работы цикла по созданным потокам. На каждом потоке происходит поиск максимального элемента, но только на определенном участке массива. Наконец, благодаря атрибуту `reduction()` с переданной аргументом функцией `max`, при выходе из параллельного блока выбирается максимальный элемент из максимальных элементов, найденных на разных потоках.

Если бы внутри параллельного блока не была объявлена директива `for`, то на каждом из потоков происходила бы одна и та же работа: находился максимальный элемент массива, причем счётчик цикла проходил бы по всем элементам массива полностью.

3. Анализ временных характеристик последовательного алгоритма

Описание эксперимента

- Измеряется время работы алгоритма на 50 различных массивах длиной 10 000 000 элементов. Находится среднее значение;
- Измеряется количество операций сравнения и присваивания во время работы алгоритма на 50 различных массивах длиной 10 000 000 элементов. Находится среднее значение.

Теоретические показатели

- Временная сложность последовательного алгоритма $O(n)$;
- Количество операций $O(n)$.

Экспериментальные показатели

- Среднее время работы алгоритма: 0.033552 [с];
- Количество операций сравнения и присваивания: 10 000 018.

4. Анализ временных характеристик параллельного алгоритма

Описание эксперимента

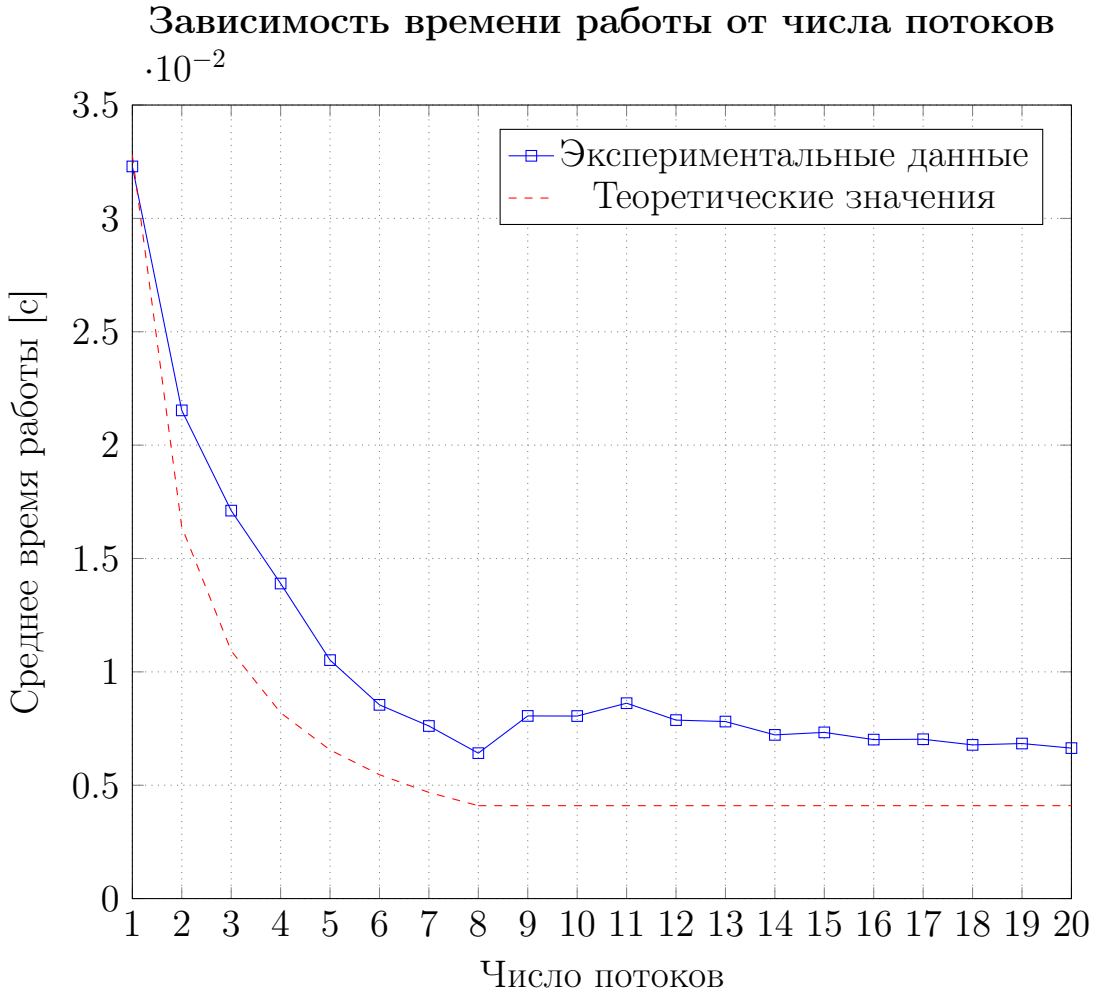
- измеряется время работы алгоритма для одного и того же массива, но на разном числе потоков: от 1 до 20;
- измерения производятся для 50 различных массивов, размер массива 10 000 000 элементов.

Результаты измерений

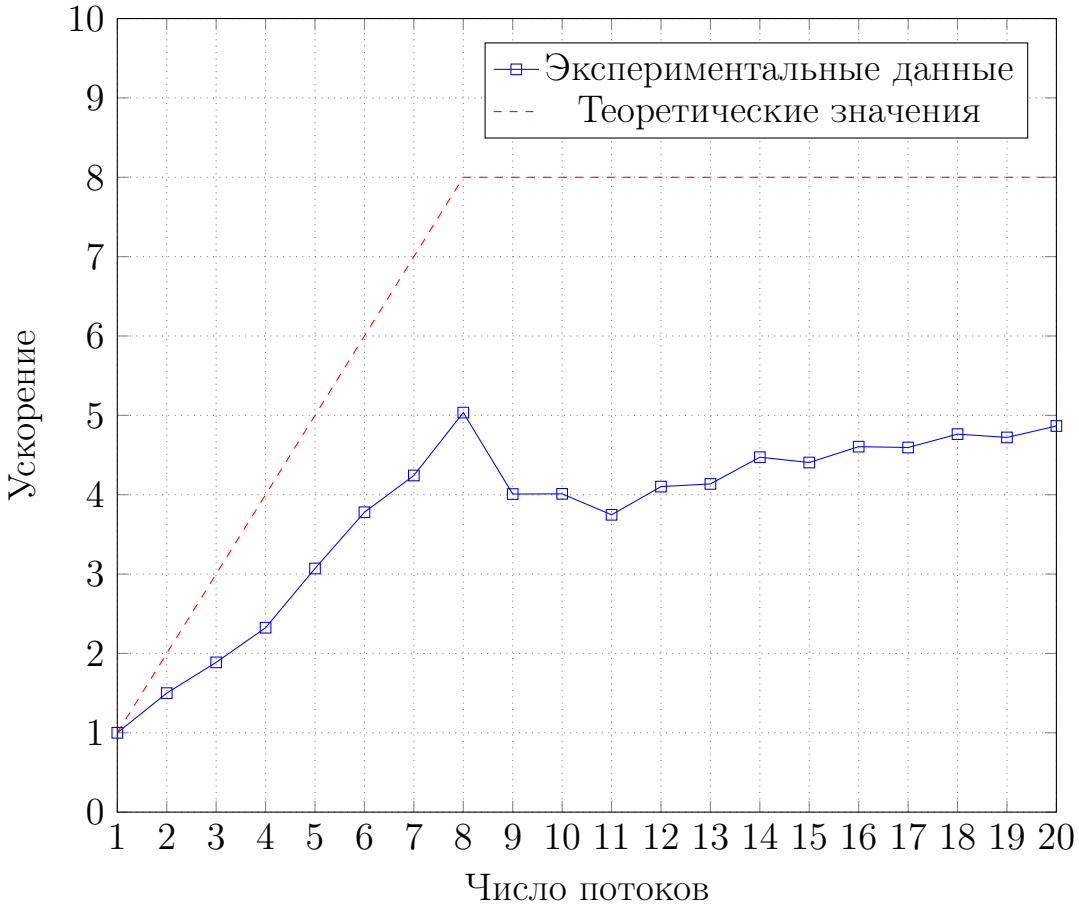
Следующая таблица содержит полученные в результате эксперимента данные: среднее время работы для различного числа потоков.

Число потоков	Среднее время работы
1	0.032292
2	0.021533
3	0.017111
4	0.013896
5	0.010518
6	0.008545
7	0.007612
8	0.006414
9	0.008056
10	0.008051
11	0.008618
12	0.007872
13	0.007808
14	0.007220
15	0.007330
16	0.007012
17	0.007028
18	0.006779
19	0.006839
20	0.006637

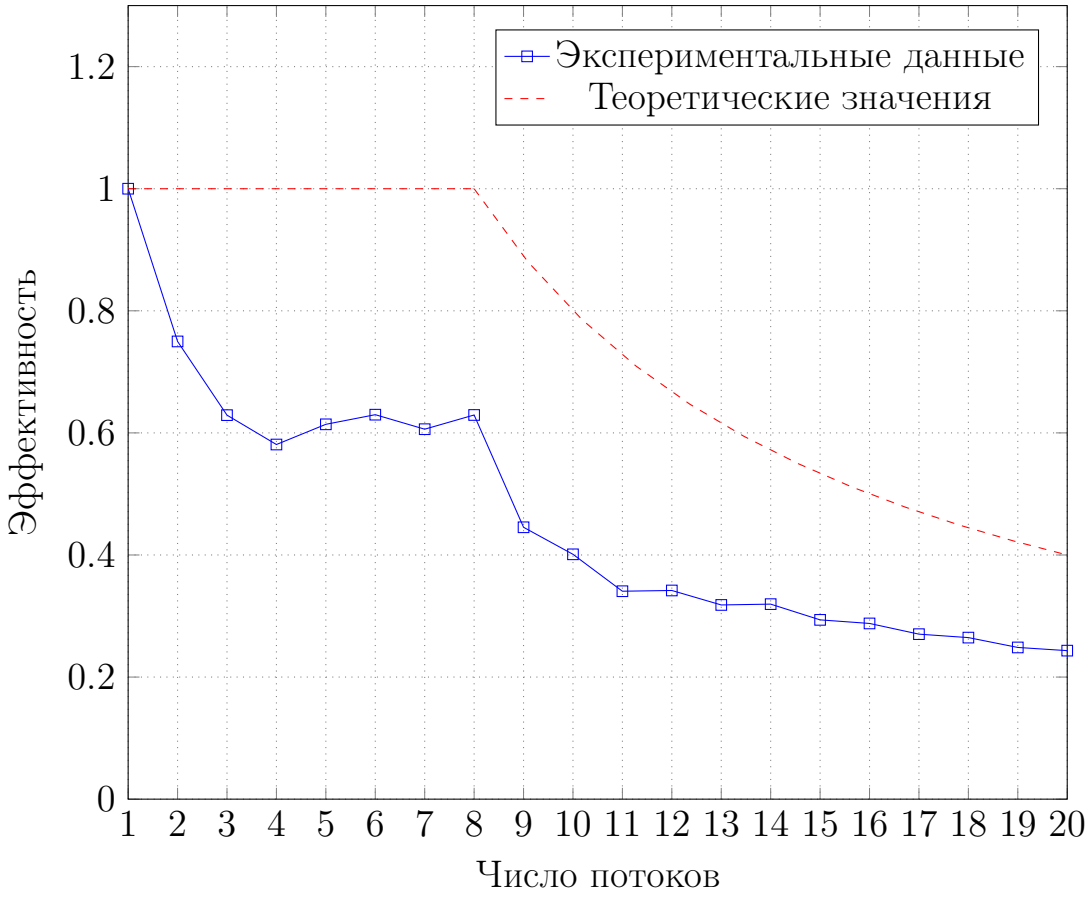
Графики



Зависимость ускорения от числа потоков



Зависимость эффективности от числа потоков



5. Заключение

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

- среднее время работы алгоритма уменьшается с ростом числа потоков до 8 включительно. Затем среднее время работы увеличивается до 0.008618с при 11 потоках. При последующем увеличении числа потоков среднее время работы уменьшается;
- минимальное время работы алгоритма составляет 0.006414с на 8 потоках;
- максимальное ускорение составляет 5.034718 на 8 потоках.

Динамика значения среднего времени работы до 8 потоков 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 9 потоках объясняется превышением требуемым числом потоков числа логических ядер.

6. Приложение

Код программы, таблицы расположены на github
Запуск программы: `make && ./run.out`