# Лабораторная работа: Параллельная обработка массива данных с использованием директивы pragma в C++

## Цель:

Научиться использовать директивы pragma для создания параллельных версий программ на C++ для эффективной обработки массивов данных.

## Задачи:

1. Реализовать последовательную версию программы на C++, которая выполняет обработку элементов массива.
2. Использовать директиву pragma для разделения обработки массива между несколькими потоками с использованием OpenMP.
3. Сравнить производительность последовательной и параллельной версий программы для различных размеров входных данных. Построить график отдельный для последовательной и параллельной версий и сравнить по графику;
4. Сделать выводы.
5. Проверить корректность работы в DEBUG/RELEASE x64;

## Описание:

Предлагается написать программу на C++, которая выполняет некоторую средне-сложную операцию (например, поиск максимального/минимального элемента и т.д. очень простые операции такие как суммирование элементов и прочее запрещено использовать) на массиве данных. Эта операция должна быть выполнена как в последовательной, так и в параллельной версиях программы с использованием директивы pragma.

Следует измерить время выполнения каждой версии программы для различных размеров входных данных, а именно для:

- 50 элементов
- 100 элементов
- 250 элементов
- 500 элементов
- 750 элементов
- 1000 элементов
- 5000 элементов
- 10000 элементов

. Результаты должны быть записаны и проанализированы.

**Подсказки к выполнению:**

- Для создания параллельной версии программы используйте библиотеку OpenMP.
- Убедитесь, что количество потоков, создаваемых параллельной версией программы, можно настраивать для достижения оптимальной производительности.
- Используйте средства для измерения времени выполнения программы, такие как std::chrono::high_resolution_clock. (*как с этим работать можно посмотреть в офиц. документации*)
- В ПРИЛОЖЕНИИ А есть примеры кода;

Подключенные библиотеки:

```
#include <iostream>

#include <vector>

#include <chrono>

#include <omp.h>
```

# ПРИЛОЖЕНИЕ А.

## 1. ПОСЛЕДОВАТЕЛЬНОЕ СУММИРОВАНИЕ ЭЛЕМЕНТОВ МАССИВ ДАННЫХ (ВЕКТОРА).

```
int sum(const std::vector<int>& array) {
    int result = 0;
    for (int i = 0; i < array.size(); ++i) {
        result += array[i];
    }
    return result;
}
```

## 2. ПАРАЛЛЕЛЬНОЕ СУММИРОВАНИЕ ЭЛЕМЕНТОВ МАССИВ ДАННЫХ (ВЕКТОРА).#1

```
int parallel_sum(const std::vector<int>& array) {
    int result = 0;
#pragma omp parallel for reduction(+:result)
    for (int i = 0; i < array.size(); ++i) {
        result += array[i];
    }
    return result;
}
```

## 3. ПАРАЛЛЕЛЬНОЕ СУММИРОВАНИЕ ЭЛЕМЕНТОВ МАССИВ ДАННЫХ (ВЕКТОРА).#2

```
int parallel_sum(const std::vector<int>& array) {
    int result = 0;
#pragma omp parallel for num_threads(4)
    for (int i = 0; i < array.size(); ++i) {
        result += array[i];
    }
    return result;
}
```

## 4. MAIN()

```
int main() {
    std::vector<int> input_sizes = {50, 100, 250};
    std::vector<double> seq_times;
    std::vector<double> par_times;

    for (int size : input_sizes) {
        std::vector<int> array(size, 1);
```

```
        auto start_seq = std::chrono::high_resolution_clock::now();
        int seq_result = sum(array);
        auto end_seq = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> seq_duration = end_seq - start_seq;
        seq_times.push_back(seq_duration.count());

        auto start_par = std::chrono::high_resolution_clock::now();
        int par_result = parallel_sum(array);
        auto end_par = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> par_duration = end_par - start_par;
        par_times.push_back(par_duration.count());

        if (seq_result != par_result) {
            std::cerr << "Ошибка: результаты последовательной и параллельной
версий не совпадают!" << std::endl;
            return 1;
        }
    }

    std::cout << "Размер данных\tВремя (последовательно)\tВремя
(параллельно)\n";
    for (int i = 0; i < input_sizes.size(); ++i) {
        std::cout << input_sizes[i] << "\t\t" << seq_times[i] << " сек\t\t\t"
<< par_times[i] << " сек\n";
    }

    return 0;
}
```

***РЕКОМЕНДОВАНО ВООБЩЕ ИСПОЛЬЗОВАТЬ НЕ СЕКУНДЫ - А МИКРОСЕКУНДЫ - РЕЗУЛЬТАТ БУДЕТ ТОЧНЕЕ;**

```
auto start_seq = std::chrono::high_resolution_clock::now();
int seq_result = sum(array);
auto end_seq = std::chrono::high_resolution_clock::now();
std::chrono::duration<double, std::micro> seq_duration = end_seq - start_seq;
seq_times.push_back(seq_duration.count());
```

**ТАКЖЕ ВАЖНО ЧТО ПРИ РАСПАРАЛЛЕЛИВАНИИ ВРЕМЯ ДОЛЖНО БЫТЬ НИЖЕ ЧЕМ ПРИ ПОСЛЕДОВАТЕЛЬНОМ!!!**