

Отчёт по лабораторной работе №2

Петрова О.А. группы ИВТ-12М

Задания 1 и 2) Проверить работу исходной программы и разработать функцию нахождения минимального значения в массиве. Вывести результат поиска минимального и максимального значений массива до и после сортировки.

Результат заданий изображён на рисунке 1.

```
Size: 10000
Before sorting:
Maximal element = 24996 has index = 5361
Minimal element = 4 has index = 1801
After sorting:
Maximal element = 24996 has index = 9999
Minimal element = 4 has index = 0
```

Рисунок 1. Результат поиска максимального и минимального значений.

Задание 3) Измерить время сортировки массива с разными размерами – 100000, 500000, 1000000. Вывести полученное время для заданных размеров.

Результат задания изображён на рисунке 2.

```
Size: 100000
Duration is: 0.1530088 seconds
Size: 500000
Duration is: 0.7230413 seconds
Size: 1000000
Duration is: 1.5460884 seconds
```

Рисунок 2. Результаты сортировки для заданных размеров массива.

Задание 4) Реализовать функцию в которой будет сравниваться заполнение векторов с помощью цикла for и цикла cilk_for. Функция должна иметь параметр sz – размер векторов. Протестировать функцию с разными размерами: 1000000, 100000, 10000, 1000, 500, 100, 50, 10.

Результат задания изображён на рисунке 3.

```

Size: 1000000
Time to fill Standard vector: 0.8100463 seconds
Time to fill Intel Magic vector: 0.3710212 seconds

Size: 100000
Time to fill Standard vector: 0.0810047 seconds
Time to fill Intel Magic vector: 0.0540031 seconds

Size: 10000
Time to fill Standard vector: 0.0080005 seconds
Time to fill Intel Magic vector: 0.0070004 seconds

Size: 1000
Time to fill Standard vector: 0.001 seconds
Time to fill Intel Magic vector: 0.001 seconds

Size: 500
Time to fill Standard vector: 0.0010001 seconds
Time to fill Intel Magic vector: 0 seconds

Size: 100
Time to fill Standard vector: 0 seconds
Time to fill Intel Magic vector: 0.001 seconds

Size: 50
Time to fill Standard vector: 0.0010001 seconds
Time to fill Intel Magic vector: 0 seconds

Size: 10
Time to fill Standard vector: 0 seconds
Time to fill Intel Magic vector: 0 seconds

```

Рисунок 3. Результаты сравнения.

Задание 5)

- Почему при небольших значениях *sz* цикл *cilk_for* уступает циклу *for* в быстродействии?

Из-за затрат времени на создание и обработку потоков.

- В каких случаях целесообразно использовать цикл *cilk_for*?

При работе с большими объёмами данных будет целесообразно его использовать. (например, массив с размером 10000 элементов и более)

- В чем принципиальное отличие параллелизации с использованием *cilk_for* от параллелизации с использованием *cilk_spawn* в паре с *cilk_sync*?

cilk_spawn – конструкция, которая может быть использована непосредственно перед вызовом функции, чтобы указать системе, что данная функция может выполняться параллельно с вызывающей.

`cilk_sync` – точка синхронизации функций. Используется, когда дальнейшие вычисления в родительской функции невозможны без результатов дочерней.

`cilk_for` – конструкция, предназначенная для распараллеливания циклов с известным количеством повторений. В процессе компиляции тело цикла конвертируется в функцию, которая вызывается рекурсивно. Планировщик автоматически распределяет поддеревья рекурсии между обработчиками.

Вывод: В данной лабораторной работе мы ознакомились с такими структурами данных, как `cilk::reducer<cilk::op_max_index<long, int>>`, `cilk::reducer<cilk::op_min_index<long,int>>`, `cilk::reducer<cilk::op_vector<int>>`. Также в результате задания 4 видно, что разница между `cilk_for` и `for` не существенная, поэтому считаю, что использовать распараллеливание для заполнения массива (вектора) - не имеет смысла.