Отчет по лабораторной работе номер два.

В работе были исследована скорость работы различных алгоритмов, реализованными с использованием различных технологий распараллеливания в .NET.

Результаты бенчмарка:

|  |
| --- |
| | Method | method | Mean | Error | StdDev |  |------------ |------------------ |-----------:|---------:|---------:|  | ParallelFor | MostCommonFile | 533.6 ms | 9.87 ms | 9.70 ms |  | Serial | MostCommonFile | 352.4 ms | 6.93 ms | 12.14 ms |  | Linq | MostCommonFile | 741.7 ms | 14.20 ms | 16.91 ms |  | Plinq | MostCommonFile | 1,112.4 ms | 32.20 ms | 83.68 ms |  | ParallelFor | MostFrequentWords | 215.8 ms | 4.24 ms | 7.65 ms |  | Serial | MostFrequentWords | 262.7 ms | 5.03 ms | 5.79 ms |  | Linq | MostFrequentWords | 466.8 ms | 9.21 ms | 19.23 ms |  | Plinq | MostFrequentWords | 289.9 ms | 5.75 ms | 12.25 ms |  | ParallelFor | MostLongWord | 110.7 ms | 2.19 ms | 2.15 ms |  | Serial | MostLongWord | 169.5 ms | 3.20 ms | 3.14 ms |  | Linq | MostLongWord | 176.5 ms | 3.25 ms | 3.04 ms |  | Plinq | MostLongWord | 233.7 ms | 4.48 ms | 5.98 ms | |

Как можно видеть, в случае задачи поиска самого длинного слова, лучше всего себя показал ParallelFor, чуть хуже последовательный алгоритм и LINQ и PLINQ в конце.

В случае с задаче поиска n самых частых слов в текстах очередность следующая:

1. ParallelFor
2. Serial
3. Plinq
4. Linq

В случае с поиском файла с наибольшим количеством слов, которые есть во всех файлах:

1. Serial
2. ParallelFor
3. Linq
4. Plinq

Linq в данном случае почти всюду показал себя не очень хорошо, в виду его декларативной структуры, мы не можем управлять каким образом он получает запрошенные данные и, судя по результатам, он выполняет лишнюю работу.

Plinq почти везде показал себя плохо, так как в целях эксперимента, я решил положиться на эвристические алгоритмы распараллеливания PLINQ, но она не смогла показать себя с лучшей стороны.

При создании алгоритмов ParallelFor, помимо использования императивного подхода, который дает контроль для избежания лишней работы, я ещё старался следить за адекватным разделением и декомпозицией при помощи Partitioner, что привело к тому, что ParallelFor показал наилучший результат.