|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Application | App-1 | App-2 | App-3 | App-4 |
| Run 1 | 0.00752 | 0.007388 | 0.122293 | 0.007539 |
| Run 2 | 0.00668 | 0.007467 | 0.115521 | 0.006271 |
| Run 3 | 0.0068 | 0.006825 | 0.112585 | 0.007069 |
| Run 4 | 0.00669 | 0.00527 | 0.114934 | 0.007077 |
| Run 5 | 0.00674 | 0.006866 | 0.120546 | 0.007076 |
| Run 6 | 0.00674 | 0.005214 | 0.111704 | 0.008195 |
| Run 7 | 0.00667 | 0.00723 | 0.111089 | 0.007196 |
| Run 8 | 0.00671 | 0.004624 | 0.10986 | 0.007531 |
| Run 9 | 0.00677 | 0.002693 | 0.114534 | 0.007131 |
| Run 10 | 0.00712 | 0.006784 | 0.111433 | 0.007703 |
| Average | 0.006845 | 0.0060361 | 0.1149599 | 0.0072788 |

Conclusion:

* Program 1 (Single Threaded) exhibits the quickest performance by operating without parallel processing, using a sole core for execution.
* Program 3 (One Thread Per Alphabet) performs the slowest due to its creation of 26 threads, resulting in increased overhead.
* Programs 2, 4, and 5 fall in between regarding their execution times.
* When choosing a multithreading approach, it's essential to weigh the advantages of parallelism against the overhead caused by thread creation. The decision should also account for the volume of sub-directories and files, influencing the most appropriate strategy.
* Embracing a moderate level of parallelism, showcased in Programs 2, 4, and 5, seems to strike a suitable balance between performance gains and the accompanying overhead in this particular scenario.