Performance Comparison

In the given implementation, pthreads have been used to paralellize the SORT and SCAN primitives.

For the first few steps, we needed to calculate two exclusive sums, for counting the number of nodes & the depth of each node. As these computations are independent, they have been calculated using separate pthreads.

Secondly, as an intermediate step, we needed to sort nodes' array with respect to their depth. For this sort, a parallel version of the merge sort was used using 4 threads. Associated input array has been split into four pieces and each piece was sorted using a separate pthread. Later these sorted pieces have been coalesced using the merge algorithm.

Finally, parent link propagation was divided into four independent chunks and each chunk was offloaded to a pthread.

Execution time by any of the implementations noted in the table is the average runtime over 25 iterations. From the table, it is observed that as input length increased performance of the parallel implementation performed better than the sequential version for all cases except for the first case. As the input size grew, the benefit of parallelization became clearer.

String Length (file name)	Execution time sequ(micro sec)	Execution time parallel(micro sec)
17 (sanity.txt)	39	492
791(small.txt)	543	241
3961 (medium.txt)	1629	959
7926(mediumPlus.txt)	4520	2277
15854(mediumPlusPlus.txt)	15353	6861
31710(almostLarge.txt)	56910	31625
87163 (large.txt)	404242	187594
174329 (veryLarge.txt)	1606321	734802
261493 (extraLarge.txt)	3508068	1612396

Table:: Execution time by each implementation(sequential & parallel) with respect to increasing input size

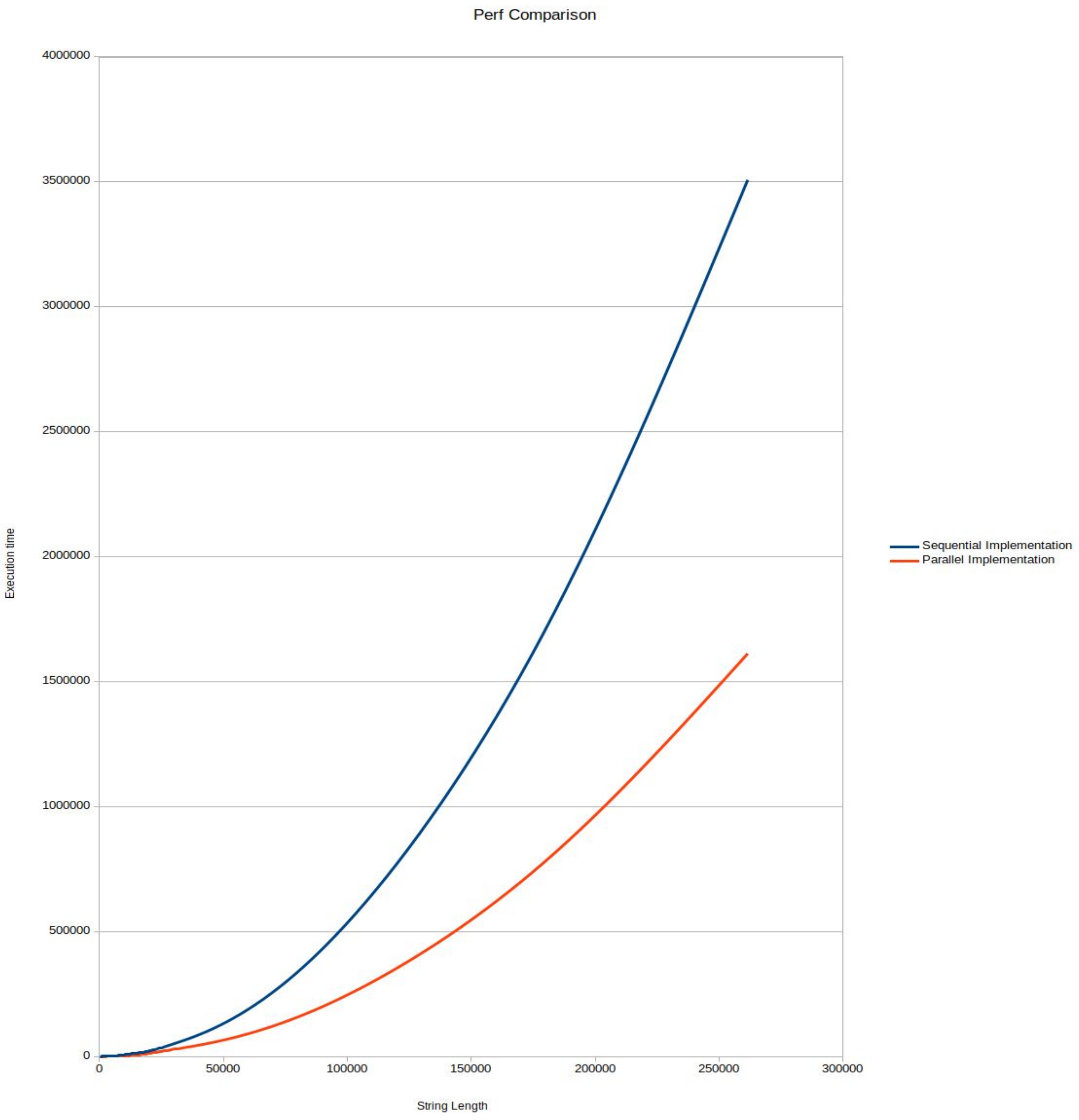

Figure: Performance comparison, Sequential vs Parallel Implementation for Dyck Language Parsing