Longest Common Subsequence in Parallel

CSE633 Parallel Algorithm - Lavanya Nadanasabapathi

University at Buffalo The State University of New York

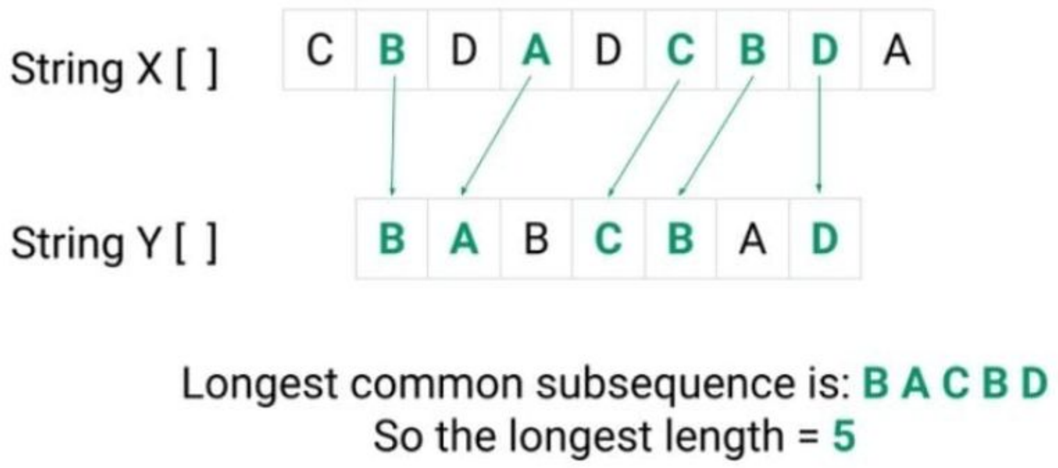
Contents

LCS Overview

The Longest Common Subsequence (LCS) problem is finding the longest subsequence present in given two sequences in the same order.

This algorithm is used in numerous fields such as bioinformatic, data mining, social networks, computer security, Git Merging etc.

Eg:



DP Approach

The dynamic programming is a classical approach for solving the LCS problem.

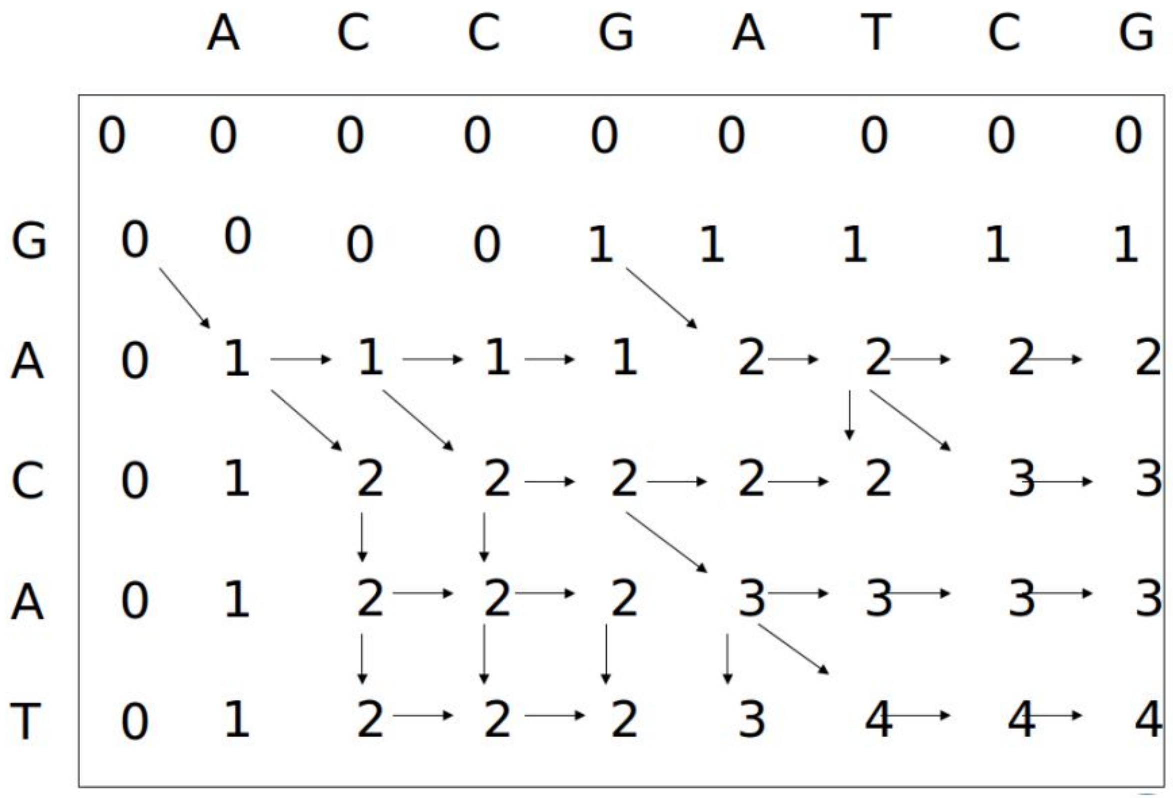
It is based on the filling of a score matrix through a scoring mechanism(a recursive formula).

$$c[i,j] = \begin{cases} 0 & \text{if } i \text{ or } j = 0 \\ c[i-1,j-1]+1 & \text{if } x_i = y_i \\ max(c[i,j-1], c[i-1,j]) & \text{if } x_i \neq y_i \end{cases}$$

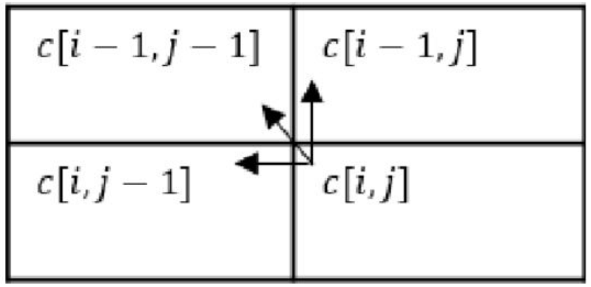
The scoring table is filled row by row using the scoring function.

The highest calculated score is the length of the LCS and the subsequence can be found by tracing back the table.

DP Approach



Time and Space Complexity: **O(mxn)**, where m and n are length of the 2 strings.



Data dependency in the score matrix

Anti-Diagonal Parallel Approach

Computing the table in **diagonal major order**, each element of the diagonal can be computed independently given that the previous diagonals are already computed.

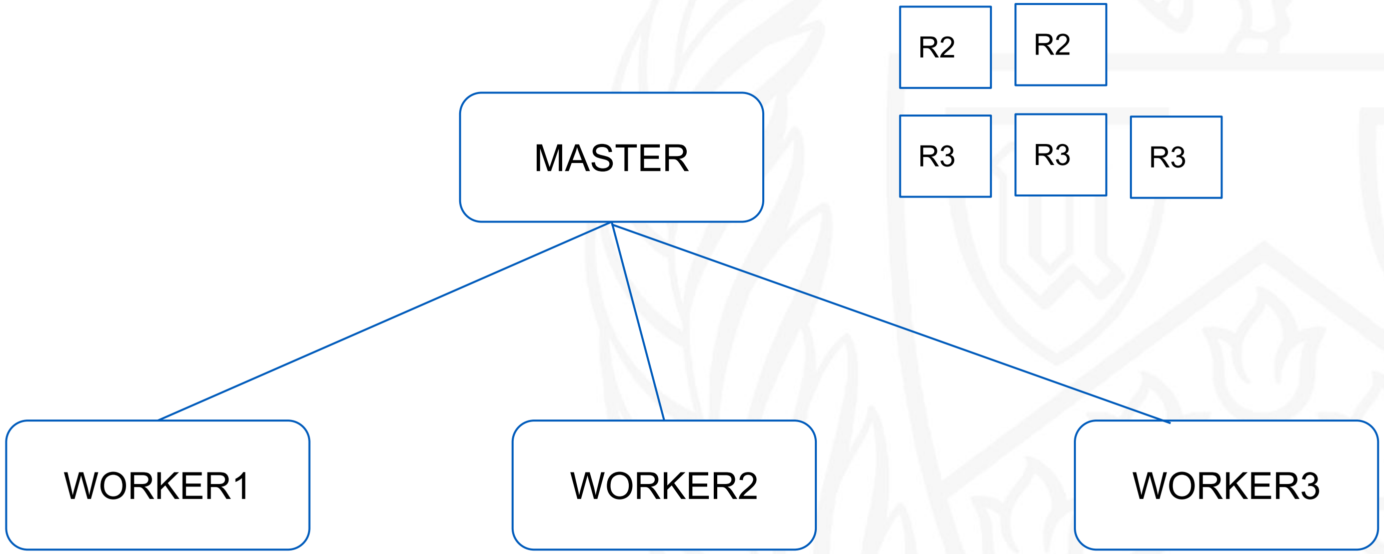
	j	0	1	2	3	4	5	6	7
i		y	A	C	B	C	B	A	B
0	x	0	0	0	0	0	0	0	0
1	B	0							
2	A	0							
3	B	0							
4	C	0							
5	A	0							
6	C	0							
7	B	0							

The value is rank of the process which is assigned to compute the value. (4 processes)

0	1	2	3	3	3	3
0	1	2	2	2	3	3
0	1	1	2	2	2	3
0	0	1	1	2	2	3
0	0	1	1	2	2	2
0	0	0	1	1	1	1
0	0	0	0	0	0	0

Anti-Diagonal Parallel Approach

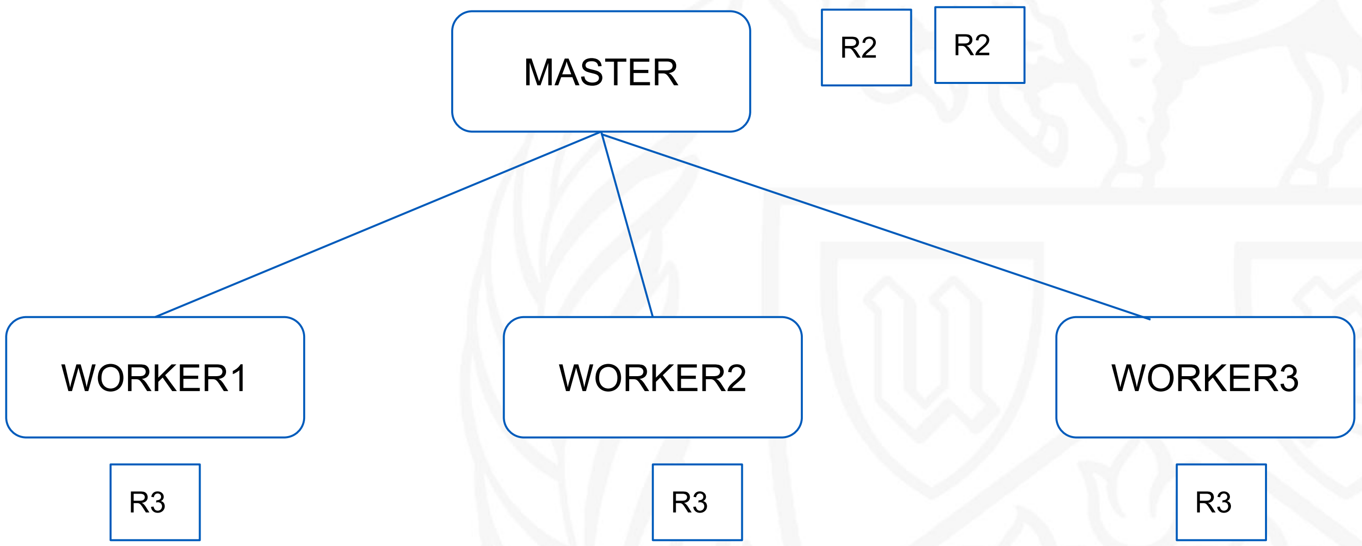
R1	R2	R3	R4	R5
R2	R3	R4	R5	R6
R3	R4	R5	R6	R7
R4	R5	R6	R7	R8



Anti-Diagonal Parallel Approach

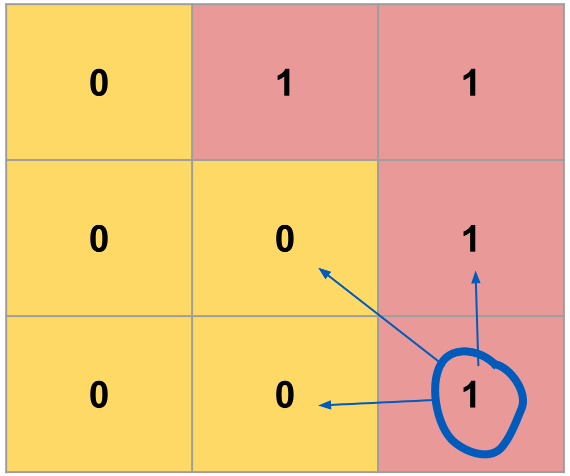
Every diagonal is computed by the N processes, with each process working on a block of data.

- The first one is to broadcast the one sequence to all nodes.
- The second part is to scatter another sequence to all nodes.
- The last one is to send/receive the completed part to other nodes. Once the data is scattered to all nodes, each node will just calculate a piece of the data instead of running the entire data in every node.



Adjacent blocks

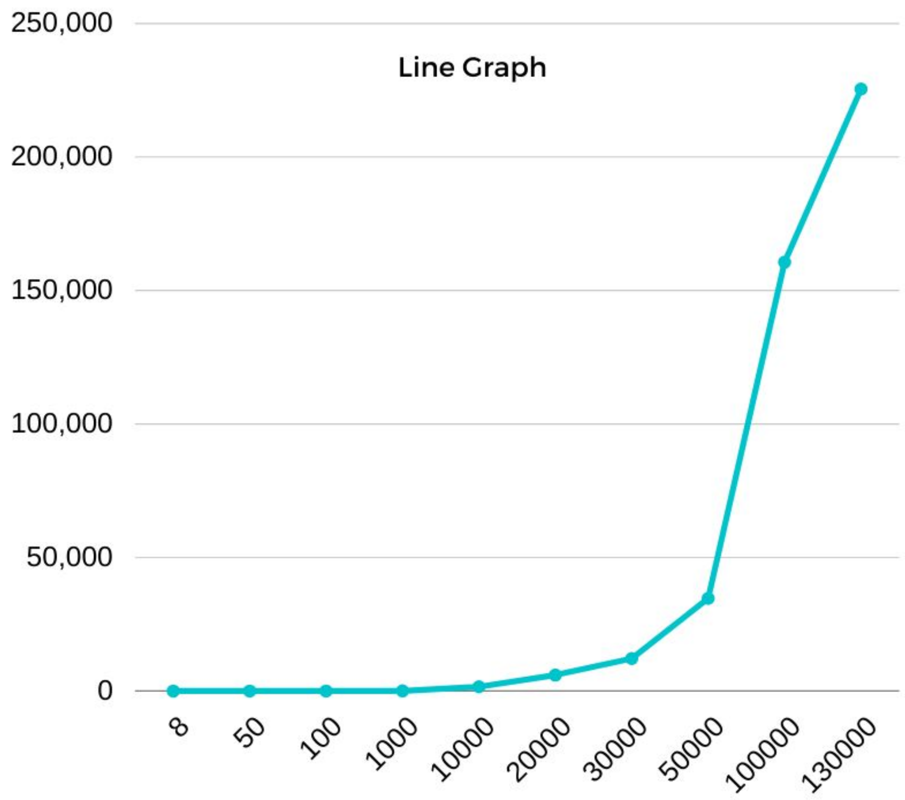
- Evaluating an entry at the boundary between two processes depend on entries of adjacent processes in previous diagonal.
- Solution: When a process computes its own block of diagonals, it receives the last element computed by the previous process and first element of following process.



Eg: Process 1 sends 1st entry of its own block to process 0 and the last element to process 2.

Result - Sequential

Input size	time (ms)
8	0.004053
50	0.089884
100	0.374079
1000	34.72805
10000	1574.15
20000	5967.81
30000	12131.86
50000	34674
100000	160606



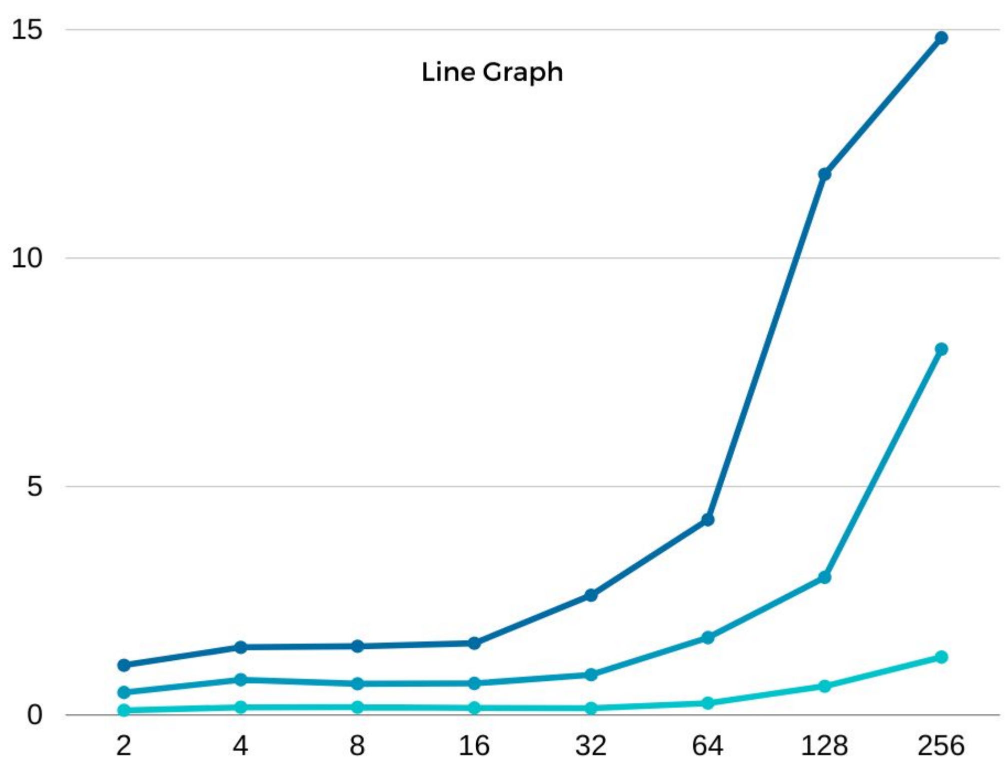
Results - Parallel

No of Processors	input size 8	input size 50	input size 100	input size 1000	input size 10000	input size 20000	input size 30000	input size 50000	input size 100000
2	0.102043	0.492	1.0881	20.85	714.703	2582.01	3543	15396.7	65245
4	0.168085	0.771	1.481	15.36	411.682	1339.26	2504	7737.12	33919
8	0.169992	0.683	1.502991	11.6	260.267	787.77	1537.35	4061.58	17338
16	0.154018	0.691	1.569986	10.85	173.125	456.61	910.85	2329.9	10216.25
32	0.146866	0.88	2.621174	12.97	131.259	299.371	614.99	1639.441	5475
64	0.258923	1.693	4.2729	29.99	236.231	543.089	857.22	2139.35	8047
128	0.628948	3.0109	11.835	41.85	429.501	812.937	1675.61	2743.01	10358
256	1.266003	8.008	14.82105	107.89	859.333	1944.154	3171.99	5667.65	15722

Results - Parallel

For small input size - Fixed Data and increase number of processors

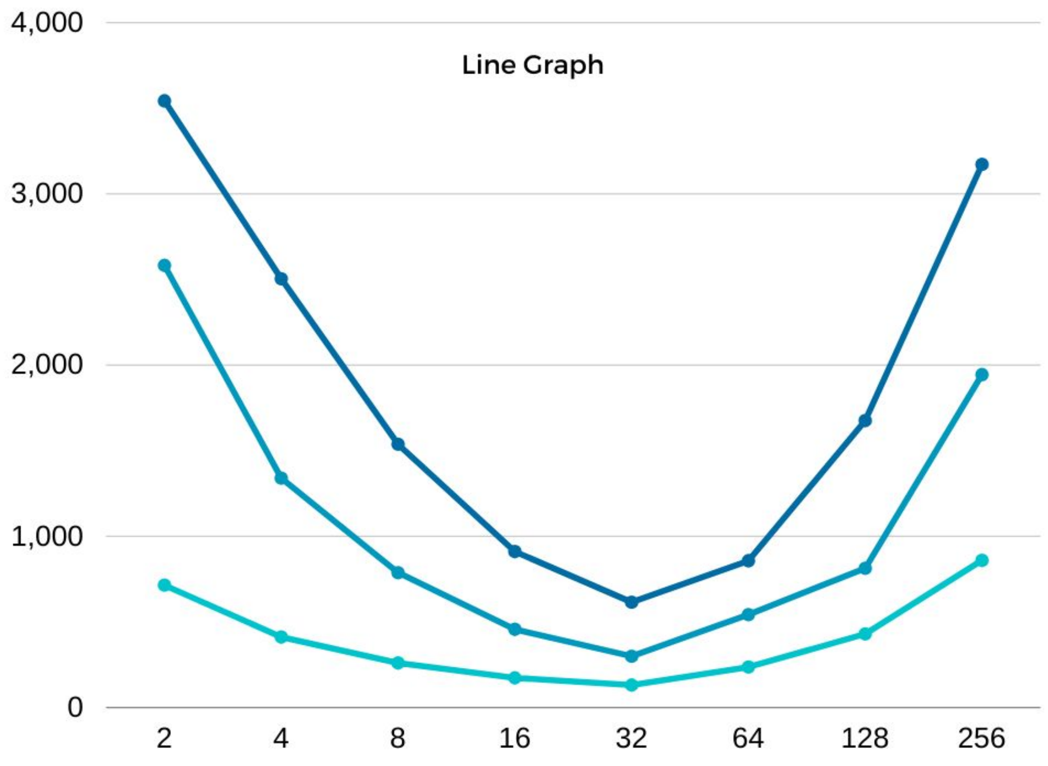
No of Processors	Input size 8	Input size 50	Input size 100
2	0.102043	0.492	1.0881
4	0.168085	0.771	1.481
8	0.169992	0.683	1.502991
16	0.154018	0.691	1.569986
32	0.146866	0.88	2.621174
64	0.258923	1.693	4.2729
128	0.628948	3.0109	11.835
256	1.266003	8.008	14.82105



Results- Parallel

For medium input size - Fixed Data and increase number of processors

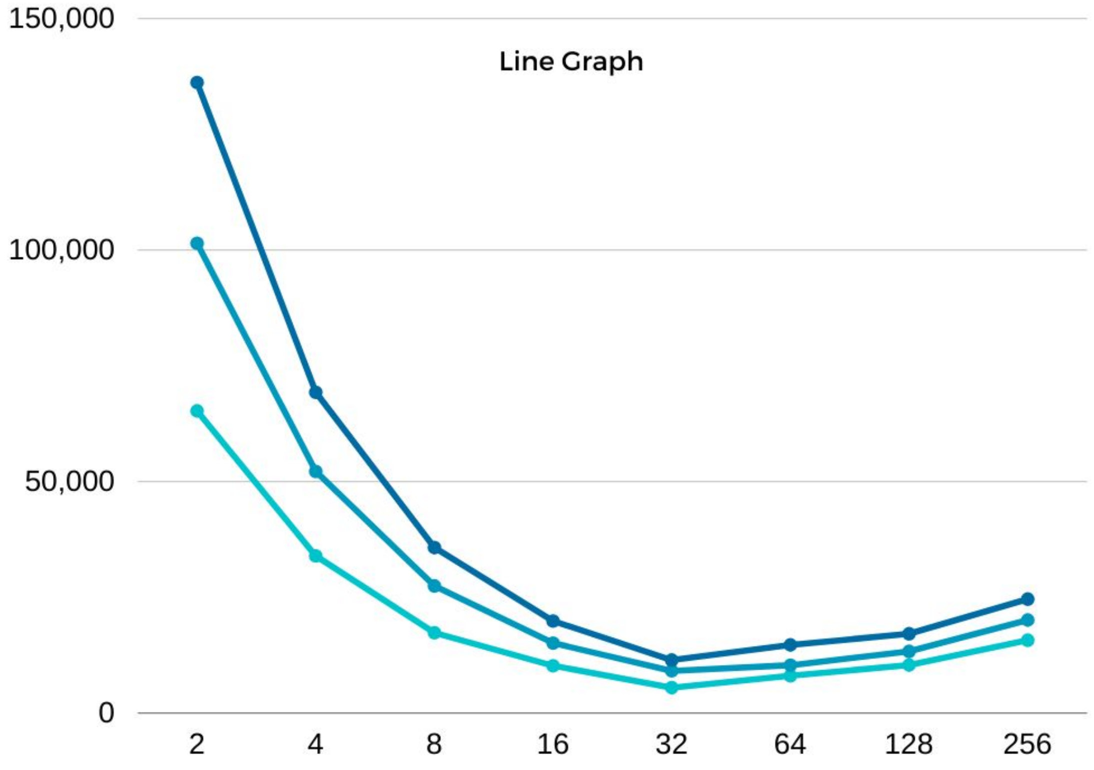
No of Processors	Input size 10000	Input size 20000	Input size 30000
2	714.703	2582.01	3543
4	411.682	1339.26	2504
8	260.267	787.77	1537.35
16	173.125	456.61	910.85
32	131.259	299.371	614.99
64	236.231	543.089	857.22
128	429.501	812.937	1675.61
256	859.333	1944.154	3171.99



Results- Parallel

For Large input size - Fixed Data and increase number of processors

No of Processors	Input size 50000	Input size 1,00,000	Input size 1,50,000
2	65245	101441	136157
4	33919	52189	69278
8	17338	27439	35726
16	10216.25	15113	19883
32	5475	9101	11433
64	8047	10323	14705
128	10358	13310	17116
256	15722	20078	24598



Observations

1. As a result of parallelization, computations are faster for large number of data.
2. As the number of processors increase, the communication overhead between the processors increase which results in longer computation time.
3. Hence selecting the optimal number of processors is critical for good computation results.

Reference

1. Zuqing Li, Aakashdeep Goyal, Haklin Kimm "Parallel Longest Common Sequence Algorithm on Multicore Systems Using OpenACC, OpenMP and OpenMPI" 2017 IEEE 11th International Symposium on Embedded Multicore/Many-core Systems-on-Chip
2. Amine Dhraief, Raik Issaoui, Abdelfettah Belghith "Parallel Computing the Longest Common Subsequence (LCS) on GPUs: Efficiency and Language Suitability" INFOCOMP 2011 : The First International Conference on Advanced Communications and Computation

Thank you!