COL380 Assignment 2
Viral Marketing

Nischay Diwan 2020CS50433
Harihar S 2020CS10878

October 2022

1 Approach for Truss Computation

To each MPI rank, we assign a subset of the vertices. We assign priority to each vertex based on it's degree and break ties according to it's value. Next, for each edge, we assign the rank which has the node of lower priority.

Next, we perform triangle enumeration. For each edge in a rank, we compute the set of triangles it belongs to. We first iterate over all edges (v, u) belonging to a rank with the node u. Then, for each node in the rank adjacent to v, we find if the edge vw exists for every edge uw in G.

Finally, we iteratively find the truss values of each edge. We initially assign support values as the number of triangles incident on it. We first fix the value of $k = 3$. Next, we delete all edges which have support less than $k - 2$. For each edge deleted, we reduce the support of edges which as a part of a triangle with it by 1. To do this, we send a message to the rank having this edge to do so. We assign the truss number of the deleted edges as $k - 1$. Finally, if we no longer have any edges with truss value less than $k - 2$, we increment the value of k by 1 and move to the next iteration.

When the verbose flag is 1, we make use of the Disjoint Set Union Structure. We iterate over values of k and then for each edge, with truss number greater than $k + 2$, we unite the components of the vertices in the edge. Here we use the same master slave communication to get the edges from other nodes to the master node. For each vertex, we find it's corresponding representative. Next, for each representative we calculate it's connected components.

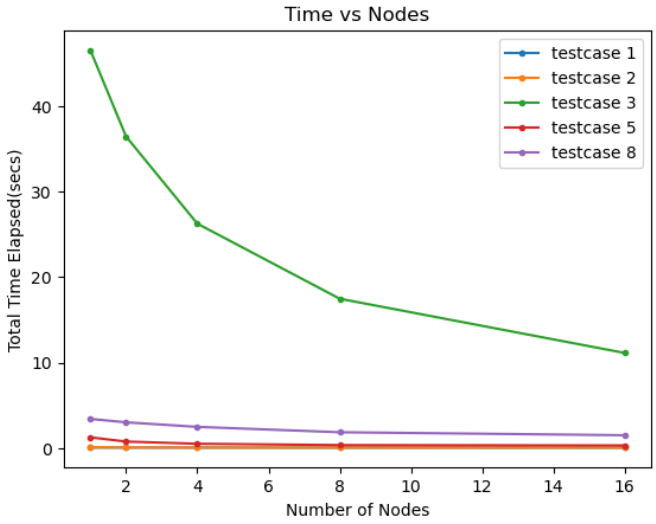
2 Approach for deadlock-free communication

Using master slave approach: First all the nodes, send their messages and destination to the master node. Master node then distribute the messages to different nodes.

3 Analysis

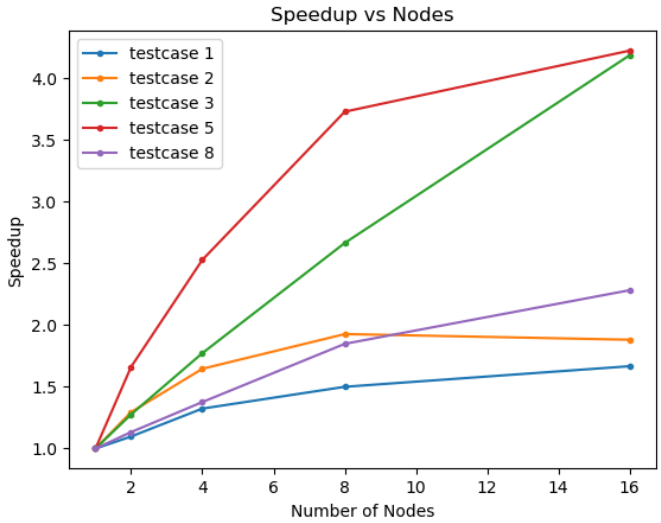
3.1 Time for each test-cases

Nodes	1	2	4	8	16
Test0	0.039	0.040	0.044	0.039	0.042
Test1	0.045	0.041	0.034	0.030	0.027
Test2	0.079	0.061	0.048	0.041	0.042
Test3	46.57	36.48	26.28	17.48	11.13
Test4	0.0017	0.0022	0.0030	0.0038	0.0042
Test5	1.241	0.747	0.491	0.333	0.294
Test6	0.91	1.04	1.03	0.85	0.71
Test7	2.80	3.44	3.81	3.46	2.99
Test8	3.40	3.00	2.47	1.84	1.49



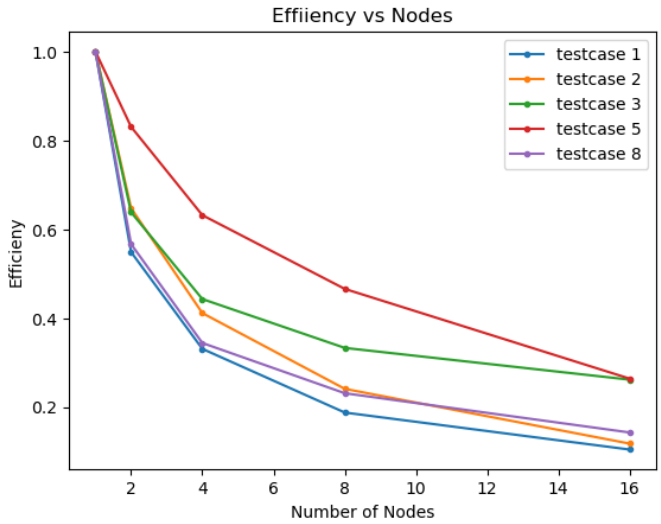
3.2 Speedup

Nodes	1	2	4	8	16
Test0	1.000	0.975	0.886	1.000	0.929
Test1	1.000	1.098	1.324	1.500	1.667
Test2	1.000	1.295	1.646	1.927	1.881
Test3	1.000	1.277	1.772	2.664	4.184
Test4	1.000	0.773	0.567	0.447	0.405
Test5	1.000	1.661	2.527	3.727	4.221
Test6	1.000	0.875	0.883	1.071	1.282
Test7	1.000	0.814	0.735	0.809	0.936
Test8	1.000	1.133	1.377	1.848	2.282



3.3 Efficiency

Nodes	1	2	4	8	16
Test0	1.000	0.487	0.222	0.125	0.058
Test1	1.000	0.549	0.331	0.188	0.104
Test2	1.000	0.648	0.411	0.241	0.118
Test3	1.000	0.638	0.443	0.333	0.262
Test4	1.000	0.386	0.142	0.056	0.025
Test5	1.000	0.831	0.632	0.466	0.264
Test6	1.000	0.438	0.221	0.134	0.080
Test7	1.000	0.407	0.184	0.101	0.059
Test8	1.000	0.567	0.344	0.231	0.143



4 Observations

We notice that out algorithm scales well with higher ranks as observed in the graphs above. On average, there is a speed up of 1.12 going from 1 rank to 2 ranks, and 1.30 speed up on going from 1 rank to 4 ranks, and 1.65 speed up on going from 1 rank to 8 ranks and finally a speed up of 2.03 on going from 1 rank to 16 ranks.