COL380 Assignment 3
Viral Marketing

Nischay Diwan 2020CS50433
Harihar S 2020CS10878

October 2022

1 Approach for Truss Computation Task 1

To each MPI rank, we assign a subset of the vertices. We assign priority to each vertex based on its degree and break ties according to its value. Next, for each edge, we assign the rank with 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 as 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 a 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 its corresponding representative. Next, for each representative, we calculate its connected components. A few of the iterations and loops are parallelized using OpenMP, but they give very minimal speedup in terms of the threads.

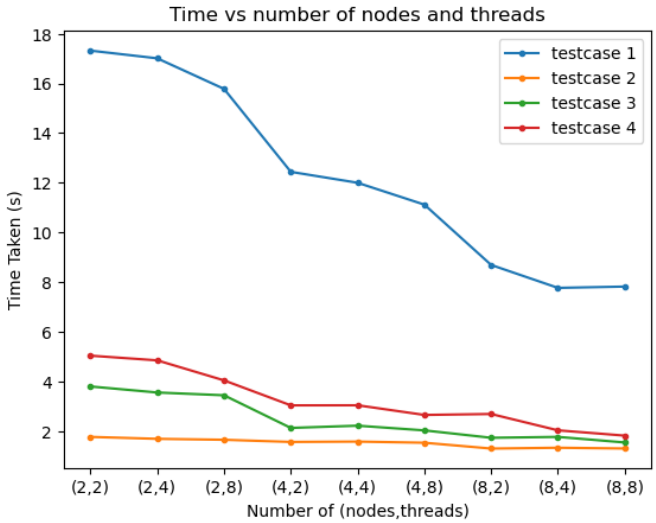
2 Approach for deadlock-free communication

Using the master-slave approach, all nodes send their messages and destination to the master node. The master node then distributes the messages to different nodes. Later for assignment 3, we tried to remove the master-slave approach, but the direct messages + synchronization overhead did not compensate for the complex method, so we removed it.

3 Analysis: Task1

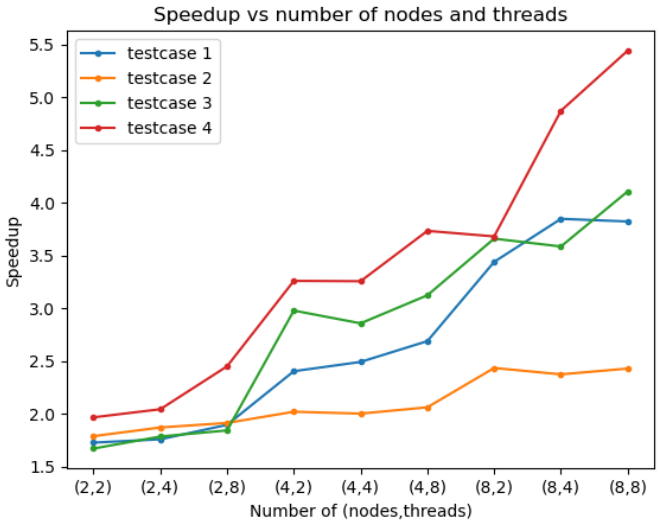
3.1 Time for each test-cases

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	17.320	17.010	15.780	12.440	12.000	11.120	8.691	7.769	7.820
Test2	1.769	1.691	1.653	1.566	1.580	1.534	1.299	1.332	1.302
Test3	3.799	3.556	3.444	2.130	2.220	2.031	1.733	1.769	1.544
Test4	5.039	4.850	4.044	3.039	3.042	2.653	2.691	2.033	1.820



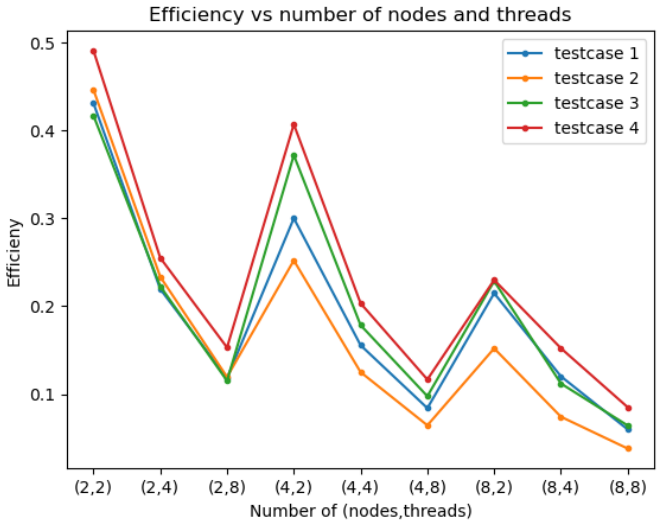
3.2 Speedup

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	1.725	1.757	1.894	2.402	2.490	2.687	3.438	3.846	3.821
Test2	1.786	1.869	1.912	2.018	2.000	2.060	2.433	2.372	2.427
Test3	1.669	1.783	1.841	2.977	2.856	3.122	3.658	3.584	4.106
Test4	1.965	2.041	2.448	3.258	3.255	3.732	3.679	4.870	5.440



3.3 Efficiency

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	0.431	0.220	0.118	0.300	0.156	0.084	0.215	0.120	0.060
Test2	0.447	0.234	0.119	0.252	0.125	0.064	0.152	0.074	0.038
Test3	0.417	0.223	0.115	0.372	0.178	0.098	0.229	0.112	0.064
Test4	0.491	0.255	0.153	0.407	0.203	0.117	0.230	0.152	0.085



4 Approach for Truss Computation Task 2

Everything followed in task 1 is used exactly the same up to the truss computation part, then the approach DSU is used similarly here but only for the endk value. Then for every node, it goes through the vertices it has. For each vertex, it goes through its neighbours and tracks the components they belong to. Now when verbose is 0, only those vertices whose components are more than p are counted, and these are gathered at rank 0, which prints it. If verbose is 1, then the vertices are stored along with the components. Again these are gathered, and all these components are printed for these vertices when influenced vertices are asked.

5 Scalabilty Analysis

We notice that our algorithm scales well with mpi processes but poorly with the OpenMP threads. This is because the min-truss algorithm we follow has a more distributed approach. So very little computation per node can be parallelized without communication and significant overhead. We can increase add parallelism to the cases where it

5.1 Iso-Efficiency

Assigning vertices to each MPI rank based on their degree. This step takes $O(n/p)$ time. For each edge, assign the rank with the node of lower priority. This step involves communication between ranks. The time taken for this step would be $O(m/p)$, assuming a uniform distribution of edges. For each edge in a rank, compute the set of triangles it belongs to. This step involves nested loops over edges and their adjacent vertices and takes $O(m/p * d^2)$ time, where d is the maximum degree of any vertex. Iteratively finding the truss values of each edge, the time taken would be $O(m^2/p^2)$. Finding the connected components of the graph. The time taken would be $O(mlogn/p)$ using a standard implementation of the disjoint set union data structure.

Overall, the time taken to solve a problem of size n using p processors can be calculated as follows:

$$T(n,p) = O(n/p + m/p + m/p * d^2 + m^2/p^2 + mlogn/p)$$

The dominant term in the above expression is m^2/p^2. Therefore, the iso-efficiency of the algorithm would be $O((nm^2)/(p^3))$, which means that the problem size can be scaled up to n = $O(p^3/m^2)$ while keeping the computation time constant. So we can take n and m to linearly increase with p.

5.2 Sequential Fraction

Calculated using Karp Flatt Metric

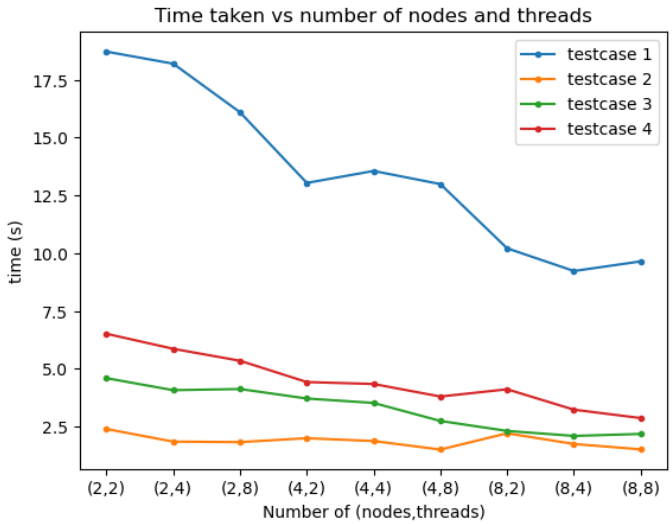
$$f = \frac{\frac{1}{S} - \frac{1}{p}}{1 - \frac{1}{p}}$$

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	0.440	0.508	0.497	0.333	0.362	0.352	0.244	0.236	0.250
Test2	0.413	0.469	0.491	0.424	0.467	0.469	0.372	0.403	0.403
Test3	0.466	0.498	0.513	0.241	0.307	0.298	0.225	0.256	0.232
Test4	0.345	0.417	0.369	0.208	0.261	0.244	0.223	0.180	0.171

6 Analysis: Task2

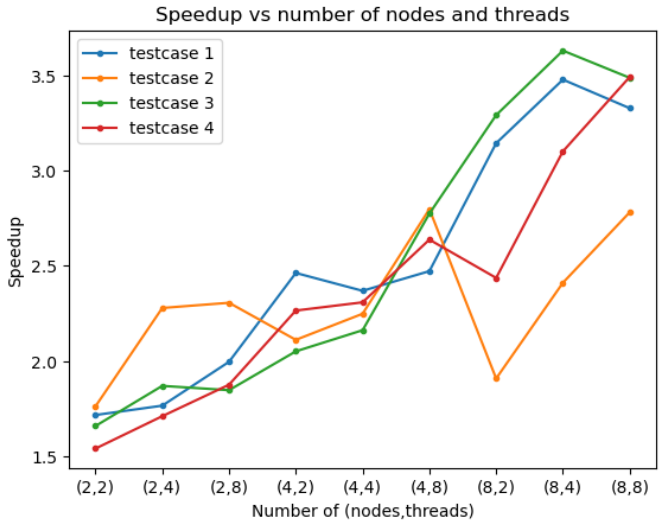
6.1 Time for each test-cases

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	18.721	18.199	16.096	13.042	13.559	12.992	10.213	9.231	9.648
Test2	2.396	1.852	1.830	1.999	1.877	1.508	2.212	1.750	1.517
Test3	4.596	4.077	4.127	3.716	3.525	2.746	2.314	2.098	2.184
Test4	6.516	5.867	5.349	4.428	4.345	3.802	4.116	3.233	2.871



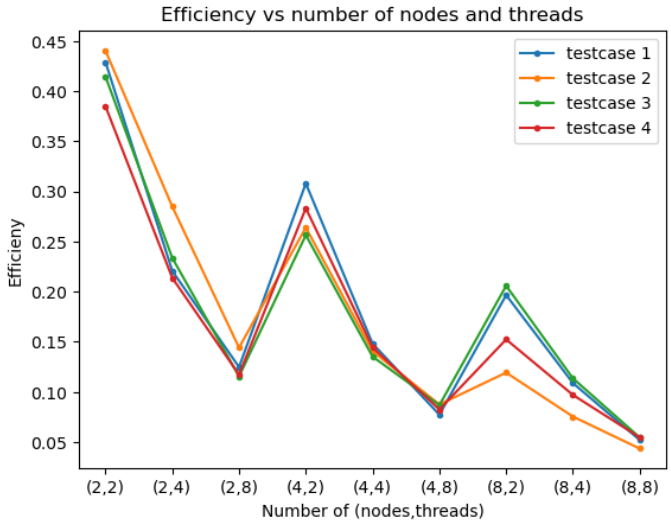
6.2 Speedup

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	1.716	1.765	1.996	2.463	2.369	2.472	3.145	3.480	3.329
Test2	1.761	2.279	2.306	2.111	2.248	2.798	1.908	2.411	2.782
Test3	1.658	1.869	1.846	2.051	2.162	2.775	3.293	3.632	3.489
Test4	1.539	1.710	1.875	2.265	2.308	2.638	2.437	3.102	3.494



6.3 Efficiency

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	0.429	0.221	0.125	0.308	0.148	0.077	0.197	0.109	0.052
Test2	0.440	0.285	0.144	0.264	0.141	0.087	0.119	0.075	0.043
Test3	0.414	0.234	0.115	0.256	0.135	0.087	0.206	0.114	0.055
Test4	0.385	0.214	0.117	0.283	0.144	0.082	0.152	0.097	0.055



7 Scalabilty Analysis

The scalability for task 2 is almost equivalent to the scalability of task 1, as the overhead is nearly the same. We don't calculate for all truss sizes but compute the DSU and go through each neighbour of every vertex. This is similar to going through all the edges since the number is distributed to all the nodes. Time taken for task 2 decreases with the number of mpi processes.

7.1 Iso-Efficiency

Here an extra overhead of going through all neighbours is added to time of task1. That will take $O(m/p)$ time. So again the iso-efficiecy remains same.

7.2 Sequential Fraction

Calculated using Karp Flatt Metric

$$f = \frac{\frac{1}{S} - \frac{1}{p}}{1 - \frac{1}{p}}$$

(Nodes,Threads)	(2,2)	(2,4)	(2,8)	(4,2)	(4,4)	(4,8)	(8,2)	(8,4)	(8,8)
Test1	0.444	0.505	0.468	0.321	0.384	0.385	0.272	0.264	0.289
Test2	0.424	0.359	0.396	0.399	0.408	0.337	0.492	0.396	0.349
Test3	0.471	0.469	0.511	0.414	0.427	0.340	0.257	0.252	0.275
Test4	0.533	0.526	0.502	0.362	0.395	0.359	0.371	0.300	0.275