COL380 ASSIGNMENT 3
ANKIT KUMAR(2020CS10323)
PARTNER WITHDRAWED THE COURSE

FOR TASK1::

1.IMPLEMENTATION

Step i) First read the values of n,m from the .gra file using MPI file reading functions.Next i read
the .dat file to find the offsets of all the nodes and use them to store their degree and neighbors from
the graph and then i distributed the graph edges among each rank that is done using graph
distribution function in main.cpp.

Step ii) From the above steps now we got our distributed graph per rank so now calculated the
support values of each edges per rank.Now in this way we have support

Step iii) Now we have a support values for the edges per rank so now i defined two scopes local Scope means per rank scope and global scope means all_rank scope. So i just implemented the mintruss function whose description is given in step iv.

Step iv) Now in mintruss truss algorithm we have to delete the edges with low support so i made two condtion one is local condition that indicates wether the edges with low support for each rank is deleted or not and second one is global condition that indicates wether the low support edges for all ranks is deleted or not. For local scope i just used a deletable stack in which i just pushed the edges which a rank have and its support is less than k-2 then in while loop until this stack get emptied i removed the edges and also reduced their support values , now there is one case also that is when deleting a edge then its neeighbor vertices might not be present in the graph of that rank as we have distributed the graph so to handle this i also used the communication function of MPI like MPI_ALLtoall, MPI_Alltoallv in this way each rank also sends Data packets(my created MPI datatype of struct variable) which has the information of the rank number which have the edge and the support value which must be deleted.In this way when each rank completes their deleting procedure my mintruss algorithm gets ended and now each rank has their own remaining supports.So to check the final value of mintruss i.e wether a group exists or not of size k-2 i used MPI_All_reduce function which takes the checks_per_rank value and performs a OR operation and returns the final value in each rank.

Step v) Now each rank gor their remaining supports and a global check value, so now for outputing the final check value to a file i used rank 0 and also for calculating the final remaining edges i made a edges_collect function which uses a my created mpi

datatype(pair) and MPI_gather function to collect all the remainig edges from each rank. now if verbose =1 then i just iterate from 1 to n and applied dfs on each node to get the path it belongs to in the remaining edges.

Step vi) To make the above algorithm better also used open mp in some fore loops in which i am reading the data from graph file and inserting to the internal data structures of my code.

2.API USED
(i)Open mpi for distributed programming.
(ii)Open mp for parallel programming.

3.RUNNING TIME TABLE FOR TASK 1

Number of processes	**Number of threads**	**Test1(4.8MB)**	**Test3(5.8MB)**	**Test4(18.9MB)**
1	1	5.226	3.056	4.234
2	2	3.621	2.301	1.856
2	4	3.562	1.193	2.924
2	8	2.416	2.294	1.836
4	2	2.565	1.985	2.385
4	4	1.468	0.866	1.274
4	8	2.585	1.972	2.438
8	2	2.16	1.825	2.084
8	4	1.003	0.69	0.975
8	8	2.11	1.8	2.113

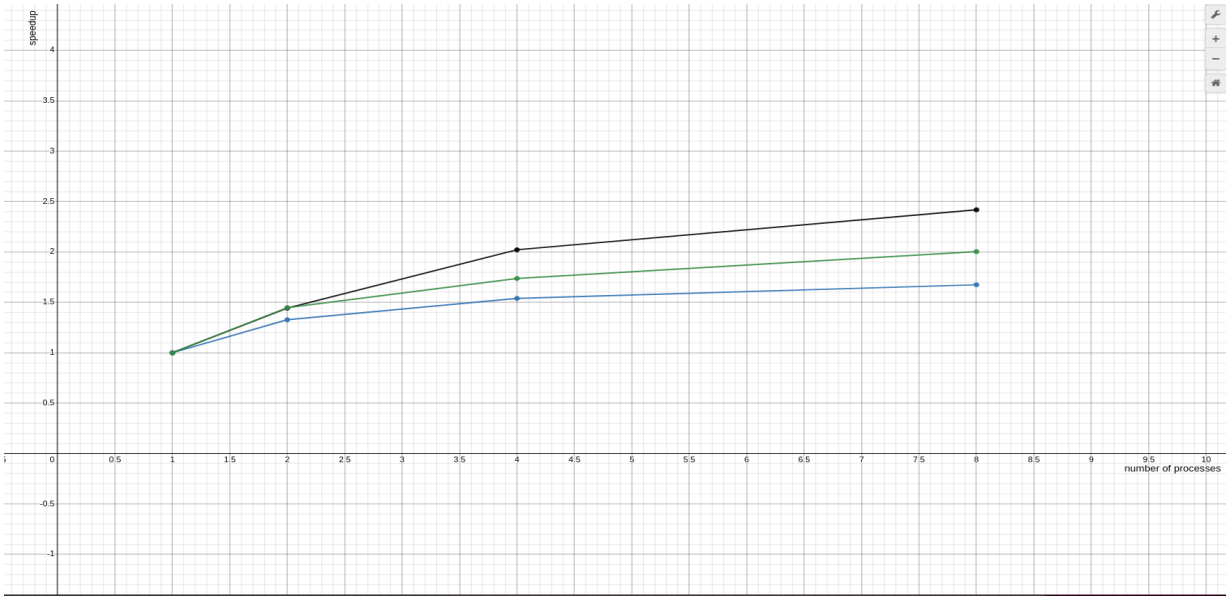
Running time(in seconds)

4.SPEEDUP TABLE FOR TASK1:

Number of processes	Number of threads	Test1(4.8MB)	Test3(5.8MB)	Test4(18.9MB)
1	1	1	1	1
2	2	1.443	1.328	2.281
2	4	1.467	2.562	1.448
2	8	2.163	1.332	2.306
4	2	2.037	1.54	1.775
4	4	3.56	3.529	3.323
4	8	2.022	1.55	1.737
8	2	2.419	1.675	2.032
8	4	5.21	4.429	4.343
8	8	2.477	1.698	2.004

Speedup for task 1

4.1 GRAPH FOR SPEEDUP TASK1:



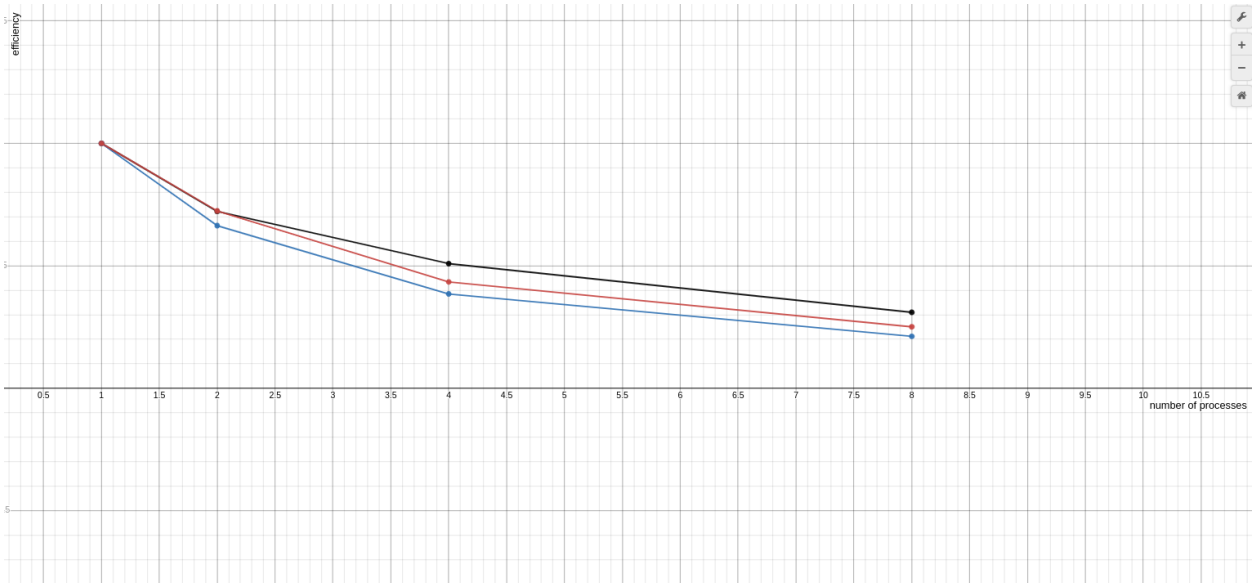
X axis-number of process , Y-speedup
black-test1 ,pink=test3,red=test4

5.EFFICIENCY TABLE FOR TASK 1:

Number of processes	Number of threads	Test1(4.8MB)	Test3(5.8MB)	Test4(18.9MB)
1	1	1	1	1
2	2	0.722	0.664	1.141
2	4	0.734	1.281	0.724
2	8	1.081	0.666	1.153
4	2	0.509	0.385	0.444
4	4	0.89	0.882	0.831
4	8	0.505	0.388	0.434
8	2	0.302	0.209	0.254
8	4	0.651	0.554	0.543
8	8	0.31	0.212	0.251

Efficiency table for task1

5.1 GRAPH FOR EFFICIENCY TASK1:



X axis-number of process , Y-efficiency
black-test1 ,pink=test3,red=test4

6.ISO EFFICIENCY FUNCTION:

The Iso-efficiency function of my algorithm came to be $I(p)=O(logp)$

Reason::

Time complexity of my algorithm is 0(n+m) where and the complexity due to all to all function is 0(log(number of processes)) if mechanism of all to all is of binary type ,here n is the number of vertices and m is the edges in the graph.

Total_time_complexity=0((M+N)*lp+logp)

Thus for the efficiency to be the same the problem must grow at the rate of O(plogp).

6. Reason for scalability of task1

Since my scalability values shows a positive trend with the number of processor and also my iso efficiency function is not so large for large set of values so we can say that my code is quite scalable.

FOR TASK2::

1.IMPLEMENTATION

Step i) First read the values of n,m from the .gra file using MPI file reading functions.Next i read

the .dat file to find the offsets of all the nodes and use them to store their degree and neighbors from
the graph and then i distributed the graph edges among each rank that is done using graph
distribution function in main.cpp.
Step ii) From the above steps now we got our distributed graph per rank so now calculated the
support values of each edges per rank.Now in this way we have support

Step iii) Now we have a support values for the edges per rank so now i defined two scopes local Scope means per rank scope and global scope means all_rank scope. So i just implemented the mintruss function whose description is given in step iv.

Step iv) Now in mintruss truss algorithm we have to delete the edges with low support so i made two condtion one is local condition that indicates wether the edges with low support for each rank is deleted or not and second one is global condition that indicates wether the low support edges for all ranks is deleted or not. For local scope i just used a deletable stack in which i just pushed the edges which a rank have and its support is less than k-2 then in while loop until this stack get emptied i removed the edges and also reduced their support values , now there is one case also that is when deleting a edge then its neeighbor vertices might not be present in the graph of that rank as we have distributed the graph so to handle this i also used the communication function of MPI like MPI_ALLtoall, MPI_Alltoallv in this way each rank also sends Data packets(my created MPI datatype of struct variable) which has the information of the rank number which have the edge and the support value which must be deleted.In this way when each rank completes their deleting procedure my mintruss algorithm gets ended and now each rank has their own remaining supports.So to check the final value of mintruss i.e wether a group exists or not of size k-2 i used MPI_All_reduce function which takes the checks_per_rank value and performs a OR operation and returns the final value in each rank.

Step v) Now each rank gor their remaining supports and a global check value, so now for outputing the final check value to a file i used rank 0 and also for calculating the final remaining edges i made a edges_collect function which uses a my created mpi datatype(pair) and MPI_gather function to collect all the remainig edges from each rank. now if verbose =1 then i just iterate from 1 to n and applied dfs on each node to get the path it belongs to in the remaining edges.

Step vi) To make the above algorithm better also used open mp in some fore loops in which i am reading the data from graph file and inserting to the internal data structures of my code.

Step vii) For task2 now when calculating the groups i had also labelled them with number to indicate that which vertex belongs to which group so that when iterating through neighbors of each vertex i easily find the influencer for verbose=0 i just outputed the count but for verbose 1 i also outputed the influencer vertices.

2.API USED
(i)Open mpi for distributed programming.
(ii)Open mp for parallel programming.

3.RUNNING TIME TABLE FOR TASK 2

Number of processes	Number of threads	Test1(4.8MB)	Test3(5.8MB)	Test4(18.9MB)
1	1	3.792	3.359	6.373
2	2	2.813	2.579	3.991
2	4	1.708	1.483	5.353
2	8	2.810	2.604	4.249
4	2	2.353	1.112	4.857
4	4	2.355	1.121	3.797
4	8	1.242	2.251	4.913
8	2	2.136	0.924	3.281

Number of processes	Number of threads	Test1(4.8MB)	Test3(5.8MB)	Test4(18.9MB)
1	1	3.792	3.359	6.373
2	2	2.813	2.579	3.991
2	4	1.708	1.483	5.353
8	4	1.025	2.129	4.543
8	8	2.116	2.069	3.450

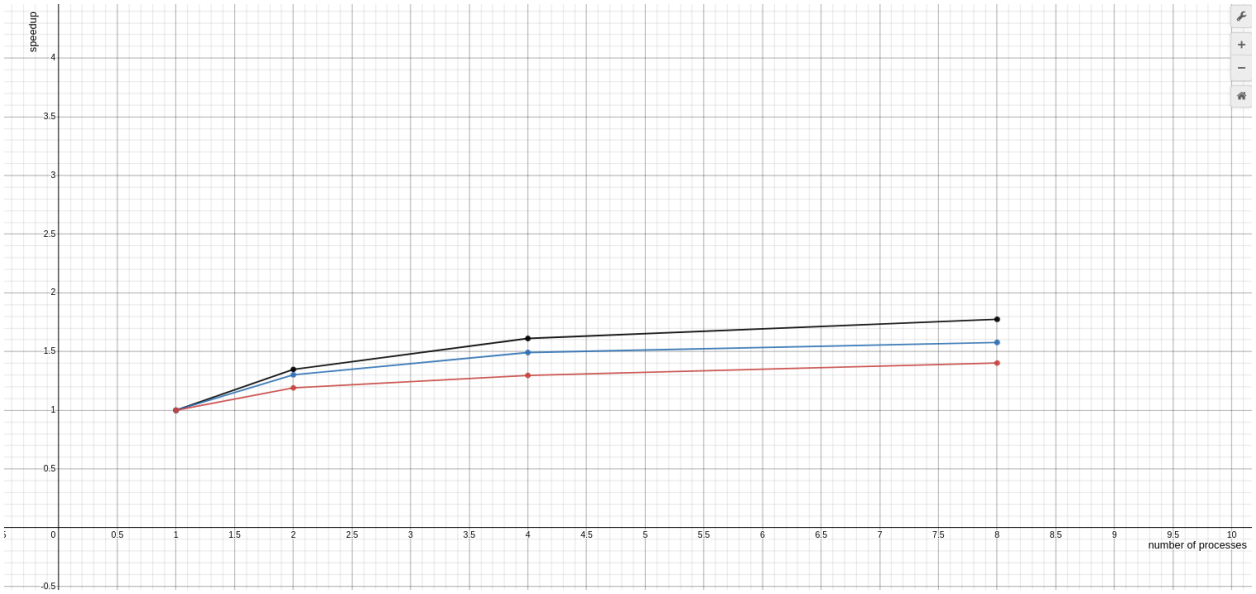
Runtime for task 2(in seconds)

4.SPEEDUP FOR TASK2

Number of processes	Number of threads	Test1(4.8MB)	Test3(5.8MB)	Test4(18.9MB)
1	1	1	1	1
2	2	1.348	1.302	1.597
2	4	2.22	2.265	1.191
2	8	1.349	1.29	1.5
4	2	1.612	3.021	1.312
4	4	1.61	2.996	1.678
4	8	3.053	1.492	1.297
8	2	1.775	3.635	1.942
8	4	3.7	1.578	1.403
8	8	1.792	1.623	1.847

Speedup for task 2

4.1 GRAPH FOR SPEEDUP TASK2:

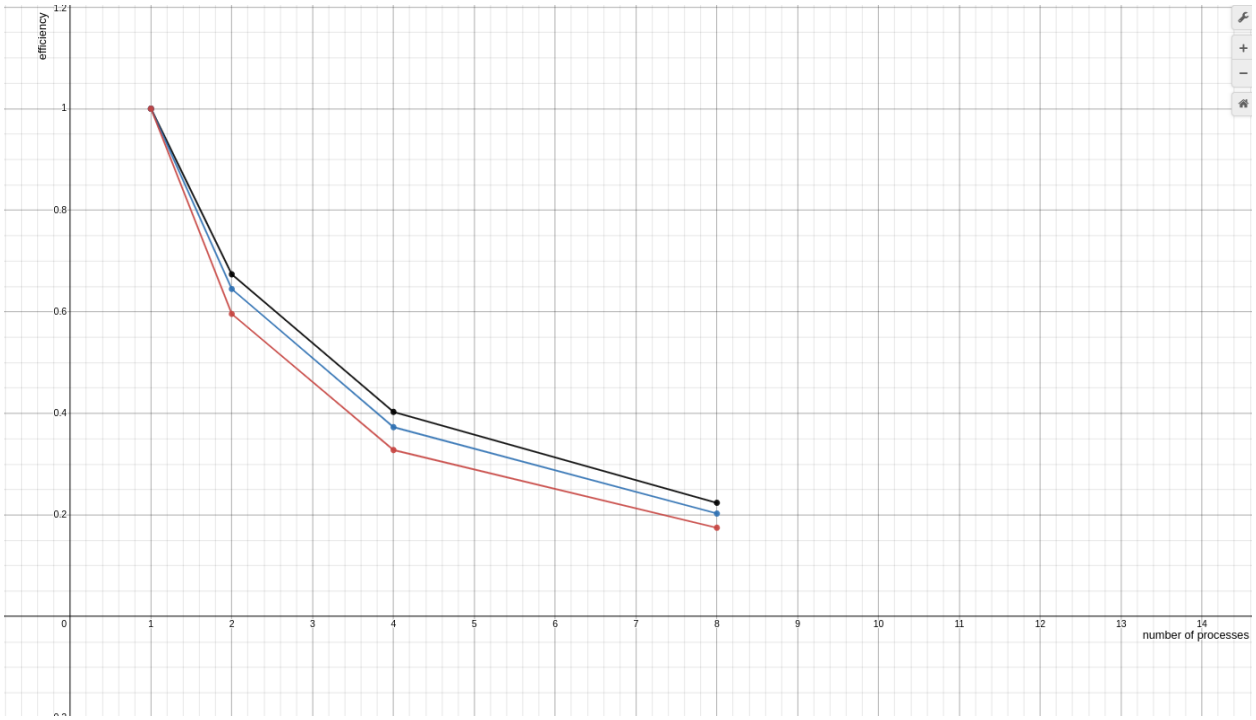


X axis-number of process , Y-speedup
black-test1 ,pink=test3,red=test4

5.EFFICIENCY TABLE FOR TASK 2:

Number of processes	Number of threads	Test1(4.8MB)	Test3(5.8MB)	Test4(18.9MB)
1	1	1	1	1
2	2	0.674	0.651	0.798
2	4	1.11	1.133	0.596
2	8	0.674	0.645	0.75
4	2	0.403	0.755	0.328
4	4	0.403	0.749	0.419
4	8	0.763	0.373	0.324
8	2	0.222	0.454	0.243
8	4	0.463	0.197	0.175
8	8	0.224	0.203	0.231

5.1 GRAPH FOR EFFICIENCY TASK2



6. Reason for scalability of task2

Since my scalability values shows a positive trend with the number of processor and also my iso efficiency function is not so large for large set of values so we can say that my code is quite scalable.