IIT DELHI

COL380: Introduction to Parallel and Distributed Programming

Assignment-3 Report

Prof. Subodh Kumar

ARYAN SANTOSH SAHOO

2019CS10335

STRUCTURE:

Initialize and Read:

First all the MPI variables are initialized, and the 8 arguments are read. The processes then proceed to construct their respective subgraphs by reading the input file and header file. I have divided the graph among all the processes in such a way that the first process receives n/p vertices the next n/p and so on and the last gets the rest. In this way we make sure that the load is evenly distributed among the processes.

Total Order:

While reading the files and constructing the graphs I have added an additional condition that only edge is read between two vertices if there exists one. And the vertex which has the lowest total order gets the edge. The total order among vertices is defined by the order that if lowest degree has lower order and if the degrees are same then the one with less value gets the lower order. I am doing this to effectively reduces the size of graph by half and to also enable easier communication among processes.

Compute Support:

I am storing my graph in the form of unordered map of unordered maps.
The support of an edge is the total number of common neighbors of both the vertices of the edge. To compute this, we need to communicate among processes, and I do this using the following method:

If all the three edges exist within the subgraph, then increment the support of all the three edges, in this case we may not have all the three vertices within the subgraph we can have two vertices but have three edges. The existence of the total order also rules out the condition that three edges exist in three different processes, this can be easily verified using the transitive condition.

A process with two edges sends request to the process with the other edge to check the existence of that edge. To find out to which process to send the request to I have a global map of vertices to the ranks of their respective process.

For communication I am sending the ordered triplet {u, v, w} which is the triangle arranged in the total order u<v<w.

If a process receives a request it sends {u, v, w, 0} if the edge doesn't exist and sends {u, v, w, 1} if the edge exists and increments the support of edge v-w.

If a process receives a response of {u, v, w, 1} it increments the support of edges u-v and u-w.

Here we are only dealing with edges and not the vertices we only need the vertex to find out which process to send request to, this is the advantage of ordering all the vertices in an order, also since my undirected graph doesn't have duplicate edges, I don't have to explicitly handle duplicate conditions.

I am using 1 MPI_Alltoall and 2 MPI_Alltoallv's for the entire communication process. First to convey the total number of requests that must be made among the processes and the other two for requests and responses.

There is significant overhead since the total number of requests that are made are in the order of O(V^3) though this is the worst case, and the average case is much smaller than this and depends on average degree and number of vertices.
This is a bottleneck for my method because I am transmitting very large numbers when using MPI_Alltoallv and the values are overflowing from INT_MAX as it is only 2*10^9, densely populated graphs or graphs with very high number of vertices will return memory faults when using my method.

This problem can be handled by breaking the large numbers or using MPI_Send and MPI_Recv for each edge for communication this method though has more total overhead.
Time complexity is O(E^2).

Compute Truss:

To calculate the truss, I have written two working functions min_truss and quick_truss. Min truss is an in-place algorithm that initializes truss numbers to support + 2 and then iteratively for the edges with the minimum truss value in the graph checks its edges with the common neighbors and if both are not settled then reduces the truss value of the edge which doesn't have the minimum truss value. After iterating over the neighbors, it settles this edge

and moves to the next edge with the minimum truss value, if all are exhausted then it enters the while loop again and this time with a new minimum truss value. This is done till all the edges are settled. Calculating minimum every time and checking settled conditions were optimized for faster computation.
Time complexity O(E^2)

The quick truss is similar to the filter edges algorithm in assignment 2.
In a queue all the edges with support value less than the k value are added then till this queue becomes empty the edges are popped and the support values of it common neighboring edges are decremented and the condition their support checked if it falls below k then it is added to the queue.
Quick truss is much faster than min truss as it deletes edges from graph.

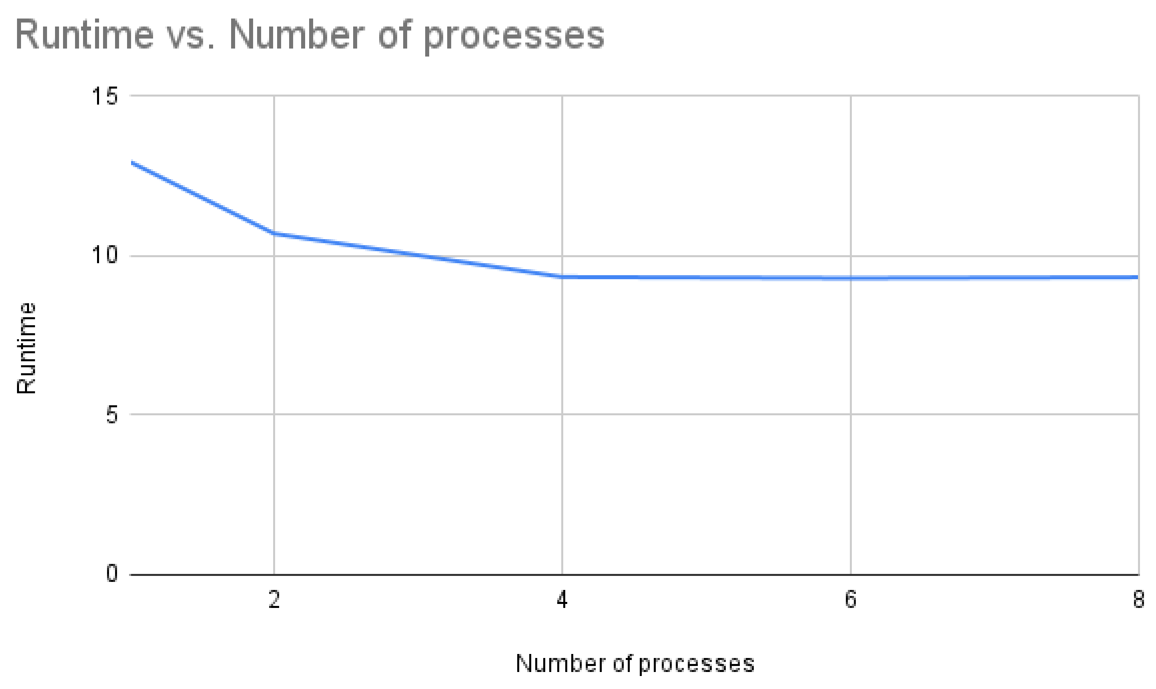
Influencer Vertices:

After computing quick truss for a given k the graph is now modified as many edges are removed. I am then calling another function on the graph which filters out isolated vertices.
Now I am using DFS to traverse the graph and find out all the connected components.
After getting all the connected components which are basically k advertisement groups, I am implementing the following algorithm to calculate the influencer vertices in the graph.

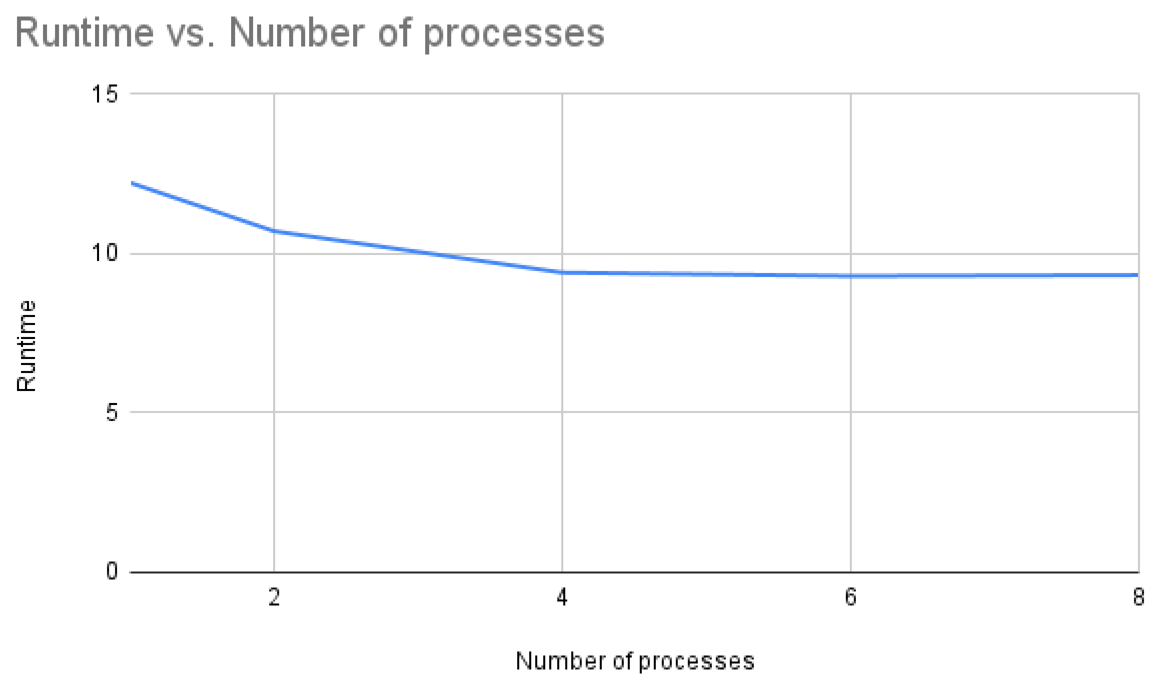
For each vertex in the original graph, I am first finding out all its neighbors then for each neighbor I am checking if it exists in any of the ktruss groups if it exists then I am adding the ktruss number of that group to an unordered set. If the size of the unordered set is greater than p, then the vertex is an influencer vertex.
Time complexity is O(V*avgDeg) or O(E).

RUNTIME(SEC) VS NUMBER OF PROCESSES FOR TEST1

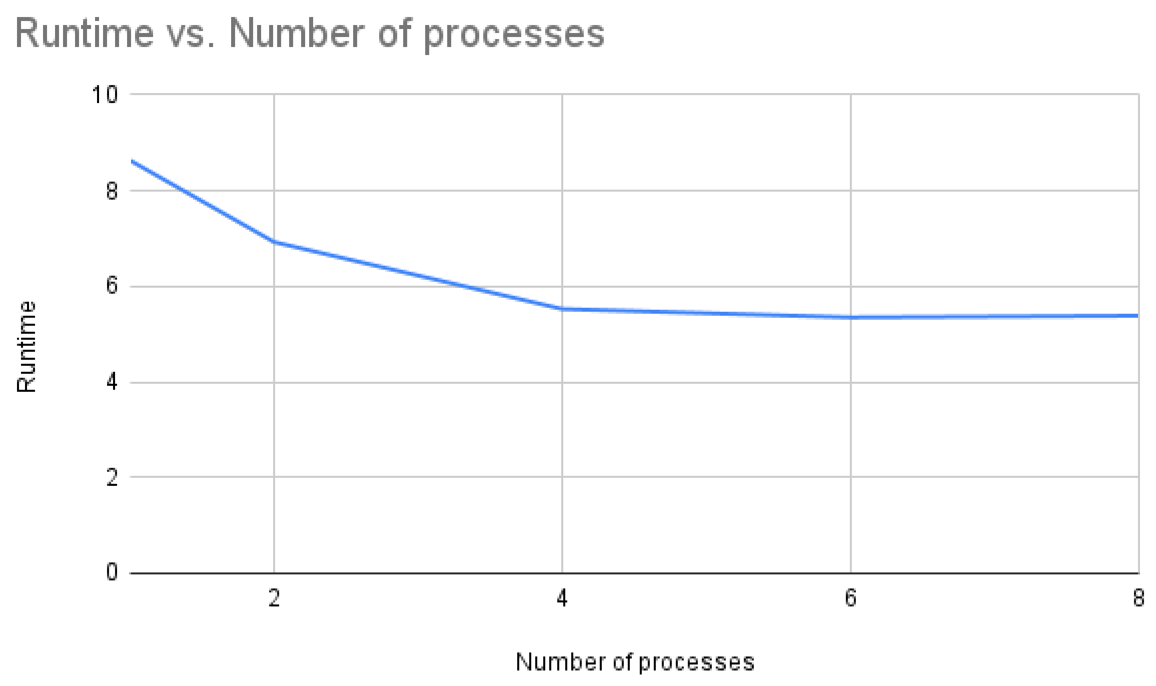
TASK = 1, VERBOSE = 0



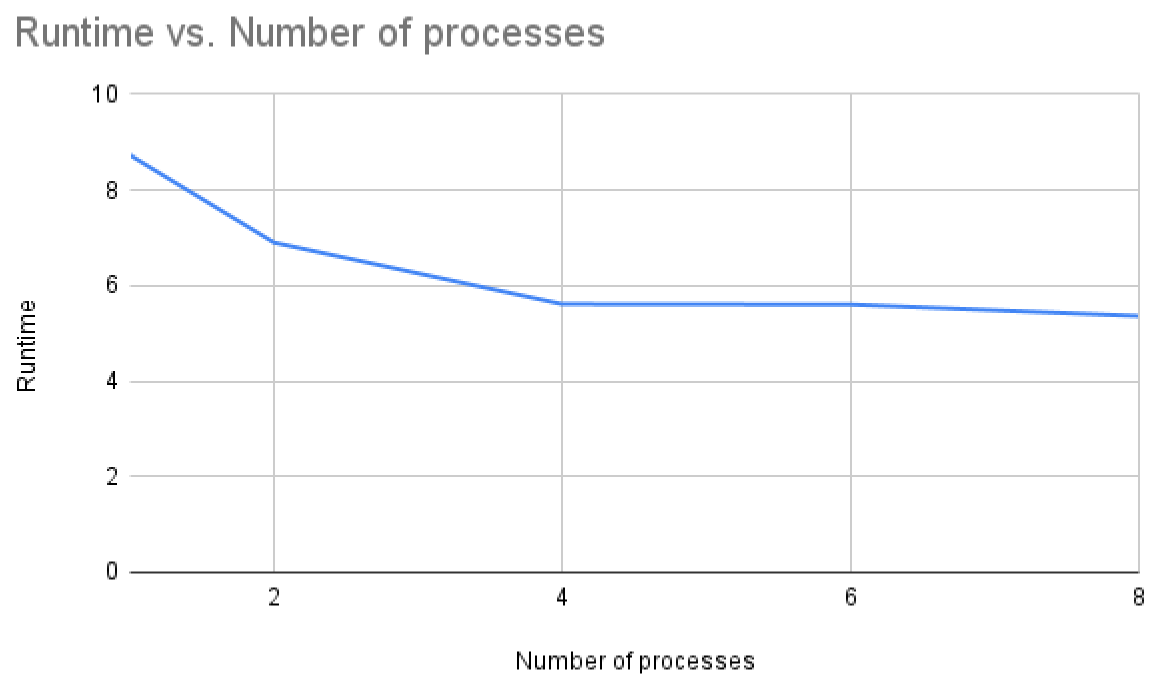
TASK = 1. VERBOSE = 1



TASK = 2, VERBOSE = 0



TASK = 2, VERBOSE = 1



ISO EFFICIENCY:

The bulk of the computing time is spent in calculating the support and the truss both of which require the most time $O(E^2)$

Since I have divided the graph among processes there is some scaling though it is poor because most of the time in these two processes goes in the communication overhead.

So its of the order:

$O(E^2/p^2) + F(V, E, p)$

The rest of the code is reading the graph calculating the number of connected components and influencer vertices.

Support and Truss:

$O(E^2/p^2) + F(V, E, p)$

Reading the graph:

Fixed time t

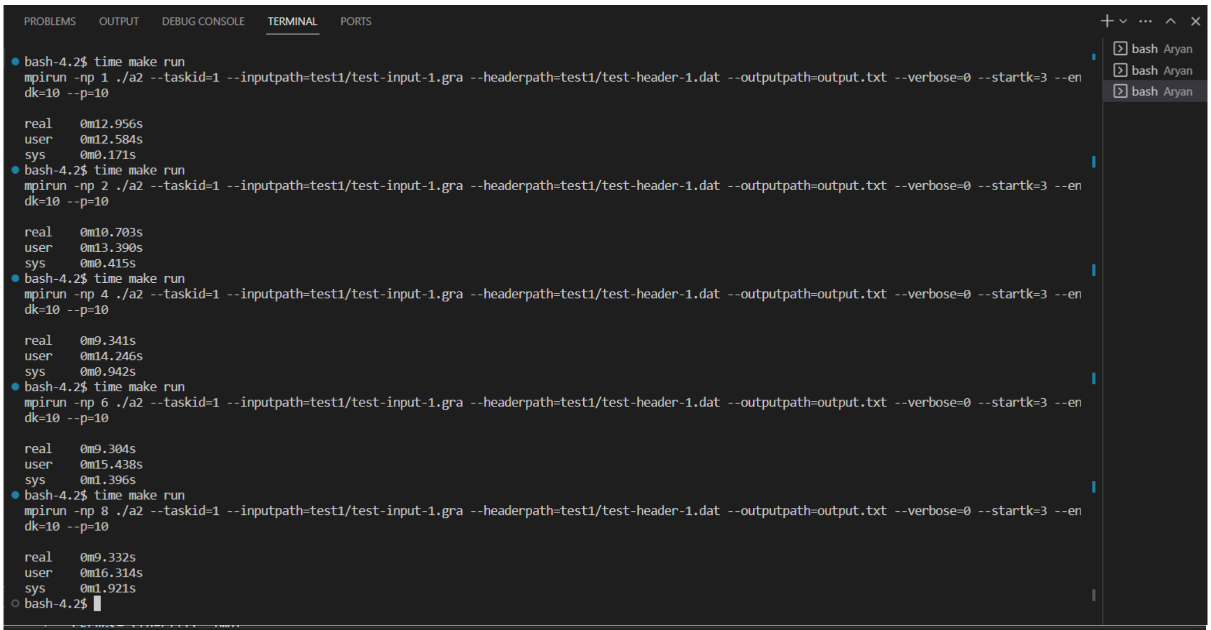
Connected Components using DFS:

$O(V + E)$

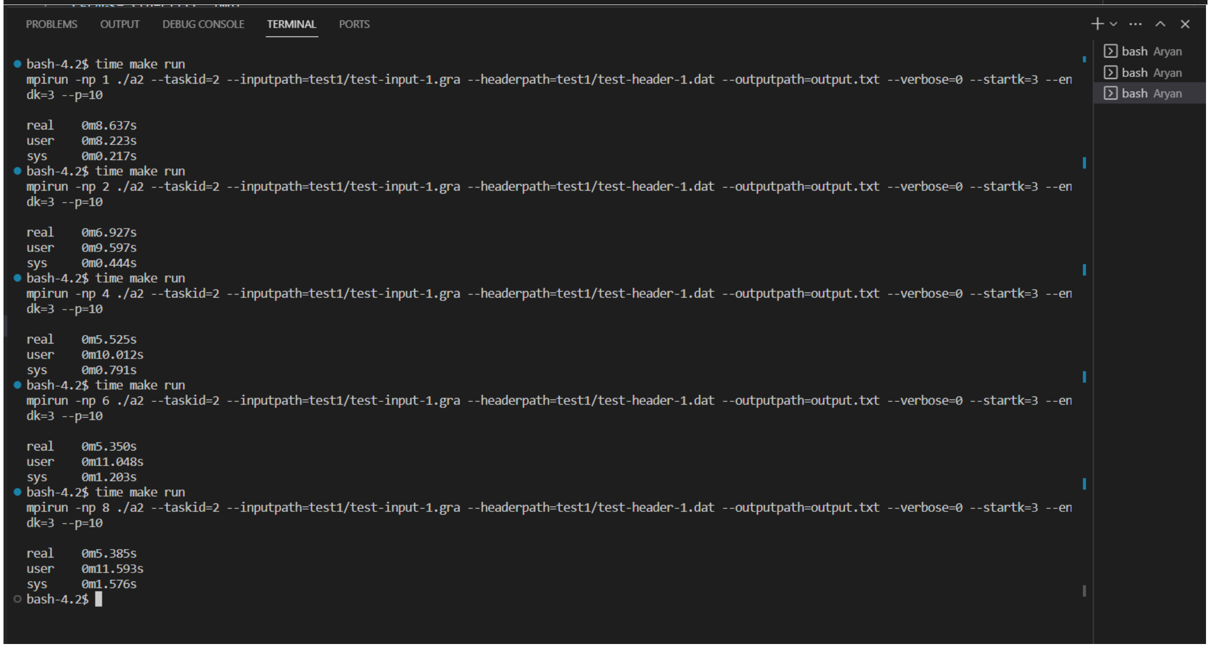
Calculating influencer vertices:

$O(V*avgDeg) = O(E)$

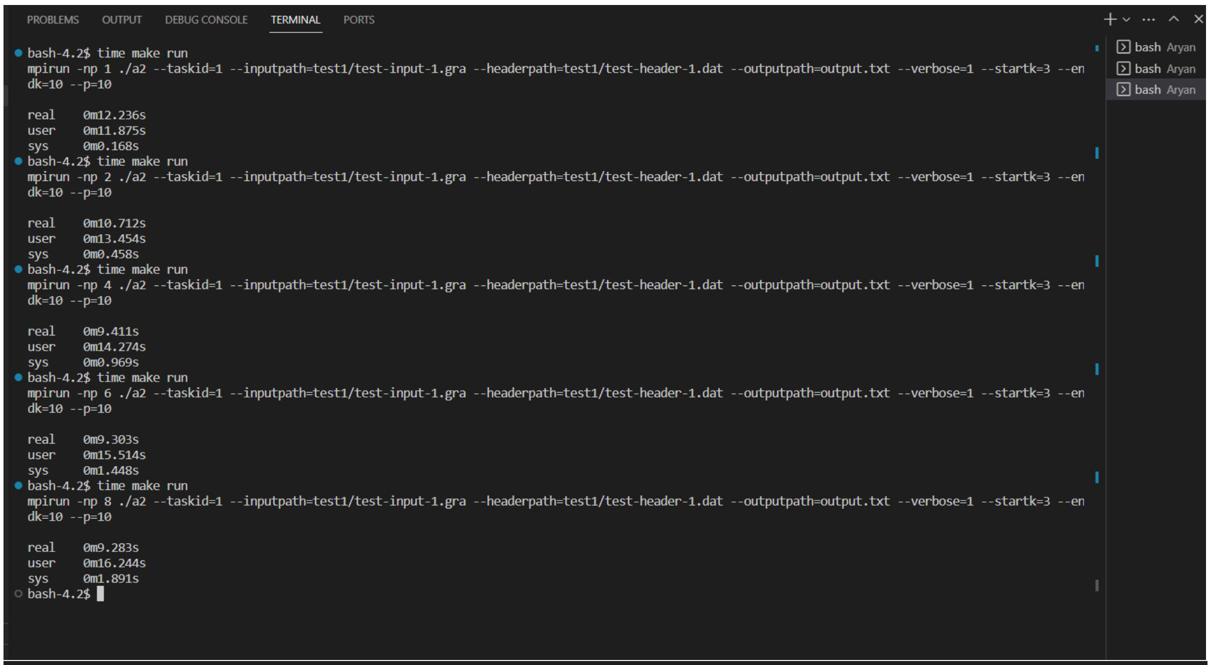
SCREENSHOTS FOR REFERENCE:



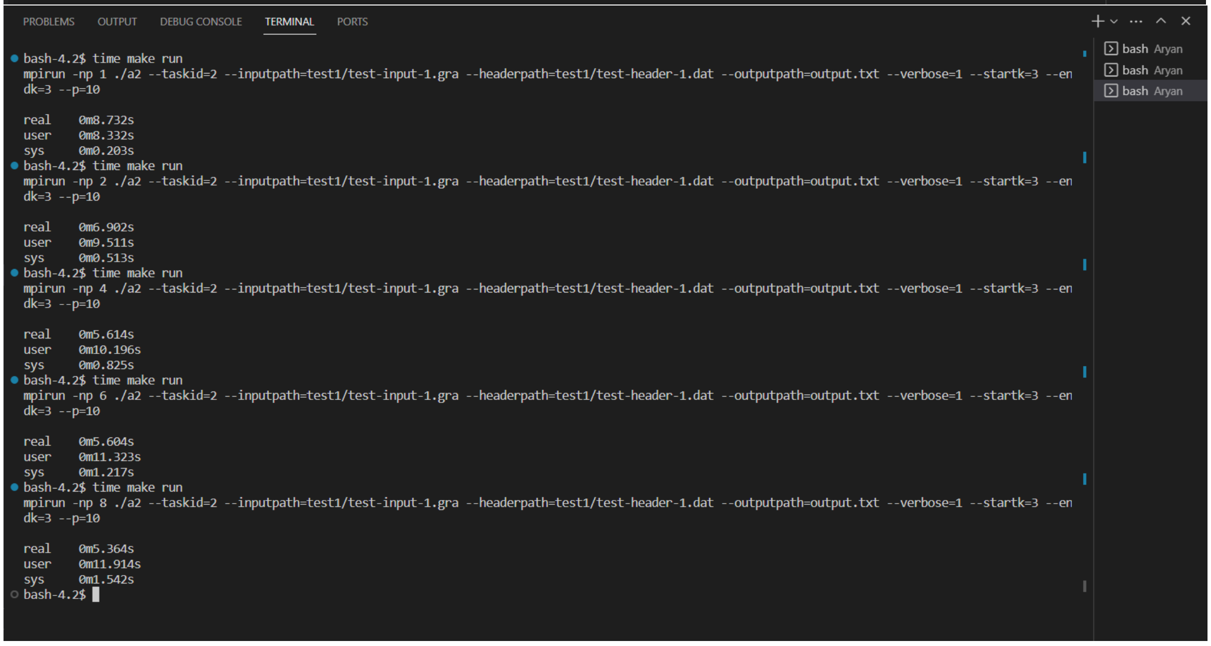
```
bash-4.2$ time make run
mpirun -np 1 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=10 --p=10

real    0m12.956s
user    0m12.584s
sys     0m0.171s
bash-4.2$ time make run
mpirun -np 2 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=10 --p=10

real    0m10.703s
user    0m13.390s
sys     0m0.415s
bash-4.2$ time make run
mpirun -np 4 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=10 --p=10

real    0m9.341s
user    0m14.246s
sys     0m0.942s
bash-4.2$ time make run
mpirun -np 6 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=10 --p=10

real    0m9.304s
user    0m15.438s
sys     0m1.396s
bash-4.2$ time make run
mpirun -np 8 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=10 --p=10

real    0m9.332s
user    0m16.314s
sys     0m1.921s
bash-4.2$
```



```
bash-4.2$ time make run
mpirun -np 1 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=3 --p=10

real    0m8.637s
user    0m8.223s
sys     0m0.217s
bash-4.2$ time make run
mpirun -np 2 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=3 --p=10

real    0m6.927s
user    0m9.597s
sys     0m0.444s
bash-4.2$ time make run
mpirun -np 4 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=3 --p=10

real    0m5.525s
user    0m10.012s
sys     0m0.791s
bash-4.2$ time make run
mpirun -np 6 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=3 --p=10

real    0m5.350s
user    0m11.048s
sys     0m1.203s
bash-4.2$ time make run
mpirun -np 8 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=0 --startk=3 --en
dk=3 --p=10

real    0m5.385s
user    0m11.593s
sys     0m1.576s
bash-4.2$
```



```
bash-4.2$ time make run
mpirun -np 1 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=10 --p=10

real    0m12.236s
user    0m11.875s
sys     0m0.168s
bash-4.2$ time make run
mpirun -np 2 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=10 --p=10

real    0m10.712s
user    0m13.454s
sys     0m0.458s
bash-4.2$ time make run
mpirun -np 4 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=10 --p=10

real    0m9.411s
user    0m14.274s
sys     0m0.969s
bash-4.2$ time make run
mpirun -np 6 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=10 --p=10

real    0m9.303s
user    0m15.514s
sys     0m1.448s
bash-4.2$ time make run
mpirun -np 8 ./a2 --taskid=1 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=10 --p=10

real    0m9.283s
user    0m16.244s
sys     0m1.891s
bash-4.2$
```



```
bash-4.2$ time make run
mpirun -np 1 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=3 --p=10

real    0m8.732s
user    0m8.332s
sys     0m0.203s
bash-4.2$ time make run
mpirun -np 2 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=3 --p=10

real    0m6.902s
user    0m9.511s
sys     0m0.513s
bash-4.2$ time make run
mpirun -np 4 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=3 --p=10

real    0m5.614s
user    0m10.196s
sys     0m0.825s
bash-4.2$ time make run
mpirun -np 6 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=3 --p=10

real    0m5.604s
user    0m11.323s
sys     0m1.217s
bash-4.2$ time make run
mpirun -np 8 ./a2 --taskid=2 --inputpath=test1/test-input-1.gra --headerpath=test1/test-header-1.dat --outputpath=output.txt --verbose=1 --startk=3 --en
dk=3 --p=10

real    0m5.364s
user    0m11.914s
sys     0m1.542s
bash-4.2$
```