This code parallelizes Dijkstra's Algorithm.

--------------------------------------------------------------------------

APPROACH -

The 2 major operations in this Algorithm are -

1) Computing the vertex with the minimum distance.

(which is being done through the function chooseVertex in the given code)

2) Calculating and updating the distances of vertex in every iteration.

(which is being done by the following line in the code -

dist[i]=min(dist[i],dist[j]+edge[j][i])

)

We have parallelized both these operations through our code.

---------------------------------------------------------------------------

TECHNIQUE -

1) For operation 1 (Choosing Minimum) , we use the following method.

We split the distance array into chunks and then send the chunks to slave processes.

The slave processes compute their local minimum and send it back to master.

The master computes its own minimum too.

Then the master compares its minimum and the minimums received from the child processes and

then computes the global minimum (the vertex with the minimum value in the complete distance array).

Hence we attain parallelism in the computation of minimum value.

2) For Operation 2 (Updation) , we use the following method:

After the master computes the global minimum  (the vertex with the minimum value in the complete distance array), it sends the value of minimum distance and also the corresponding row in the edge matrix (which is the adjacency matrix for our given graph).

These values are received parallel to all the slave processes and they simultaneously update their individual chunks of thee distance array.

Steps / Pseudocode -

Code for Master -

 Create the adjacency matrix for the graph (The name of this matrix is edge[][] ).

 Create the distance array. This is called dist[].

 Break dist[] into chunks and then send the corresponding chunks to slaves.

 Create another array found[] which is have all zero values initially.

 Create another variable count and set count=1.

 Set found[source]=1 . Set count = 1.

 while(count < number of vertices)

 {

 Break the found[] into chunks and then send those chunks to slaves.

 Compute the minimum of the distance array of the master's chunks.

 Get the local minimum from all other slaves.

 Compute the global minimum by choosing the minimum out of all the local minimums and its own minimum.

 Let this index of the minimum distance be j.

 Set count =count+1 and update found[j]=1.

 Send this updated value of count to all the slaves.

 Send the dist[j] and the row edge[] to all the slaves.

 Update its own copy of dist[] as per the global minimum.

 }//end of while

Code for Slaves -

 Receive the chunk of the dist array.

 Initialize count=1.

 while(count < number of vertices)

 {

 Receive the chunk of the found array.

 Calculate local minimum and send to master.

 Receive dist[j] and the row edge[j] from the master.

 Update the dist[] chunk accordingly.

 Receive the updated value of count from master .

 }