ABSTRACT

A graph is a set of vertices and edges; a vertex may present a state or a condition while the edges may present a relation between two vertices. There is some structural change of the computer architecture to increase the power of computing will finally reshape algorithmic graph theory. In this case, parallel computing is one of the popular new computing architectures. That give some exciting results in graph theory. We will apply the same data with sequential and parallel computing with the Kruskal algorithm. As the behavior of parallel approach, we worked with a large graph then we got the difference between those two computations time.

1. Introduction:

Parallel computing is very different from sequential computing as it has different computing models other sense, the multiprocessor structure. From some point, the parallel algorithm design is flexible when the algorithms complexity increase. Basically, parallel computing was born with graph theory related. Taking the advantages from multiprocessor the basic idea of parallel computing came. Graph theory is the base of parallel computing. We get lots of NP-hard, NP-complete problems from graph theory like Travers salesman problem, Hamiltonian path. On the contrary parallel computing guided to us because computer scientist identified that this would be a good way to give extra computing powers. People solve those hard graph problems through parallel computing.

There are few parallel graph algorithms like prim’s algorithms, Kruskal’s algorithms. Kruskal’s algorithm performs better in typical situations (sparse graphs) because it uses simpler data structures. Prim’s Algorithm is significantly faster in the limit when you’ve got a dense graph with many more edges than vertices.

2. Related work:

Many researchers have worked on parallel calculation. [3] in this paper they proposed efficient technique for parallel manipulation of data structures that avoids memory access conflicts. That is, this technique works on the Exclusive Read/Exclusive Write (ER EW) model of computation, which is the weakest shared memory, MIMD machine model. [2] where for finding a minimum spanning tree the graph was divided to smaller parts and sent as an input to multiple nodes where the independent Burovka’s
The results are merged by hybrid merging.

3. Implementation:
In our report we use Kruskal algorithm for parallel calculation to find minimum spanning tree. Before going to use this algorithm, we are going to learn how does it work and its definition.

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest. It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The difference from Prim’s is that in Kruskal’s, the edge doesn’t need to be connected to a particular subtree; instead, it is chosen from the set of all edges. If we were to visualize this, it would look like increasingly large segments are drawn onto the graph, until at the last step every node is connected. At each step, we would see multiple smaller trees get connected to form a larger tree.

Algorithm for The Kruskal Algorithm.

KRUSKAL(G):

```
A = Ø
For each vertex v ∈ G.V:
    MAKE-SET(v)
For each edge (u, v) ∈ G.E ordered by increasing order by weight (u, v):
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A ∪ {(u, v)}
        UNION (u, v)
return A
```

4. Proposed work:
We are applying parallel calculation for finding a minimum spanning tree by Kruskal algorithm. Here we used the data decomposition technique for parallelizing the Kruskal algorithm. We write serial and parallel code by python programming language. Evaluation for the serial and parallel code be like this, the time given by the serial code execution is less than the time given by the parallel calculation.

5. Result and Discussion:
I used python programming language for the programming part. My computer is core i5, with 16GB RAM.
We used same data for both serial and parallel Kruskal algorithm and the experimental result shows that serial code execution time is lower than the parallel code execution. The numerical
difference is very less but still it is showing that serial is faster than the serial one. Here the problem is occurring because I used small number of nodes. The difference is very less but still it’s not showing the result we wanted. In the case of 7 vertex graph the execution time for serial is 0.016220092773 and the execution time for parallel calculation is 0.0231168270111. But when the numbers of node increased like 500 nodes then we can see parallel one works faster than the serial one.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Serial</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.01622s</td>
<td>0.0231s</td>
</tr>
<tr>
<td>500</td>
<td>2.1585s</td>
<td>1.7585s</td>
</tr>
</tbody>
</table>

6. Conclusion:
Parallel calculation is now not always giving us the small execution time. Basically, it depends on which type of work and which type of data we are using. If we use very small amount of data, then it could be possible that serial work fast than the parallel calculation. When we use big amount of data then it’s exactly shows that parallel work better than serial.
In our work we try the number of nodes like 7 and 500 but still we can do use huge number of data. So, if we use huge number of data in future then we can get more better result. Then we can use some high-speed computer like HPC of University of Tartu.

References: