Assignment No. 1

Title : Parallel Breadth First Search and Depth First Search.

Problem Statement: To implement Parallel Bredth First Search based on existing algorithms using OPENMP. Use a Tree or an undirected graph for BFS and DFS.

Objective : To implement parallel Breadth First Search and Depth First Search.

Hardware and Software Required:

Theory :

a) BFS ~~and~~ or (Breadth First search):

It is a graph traversal algorithm that explores a graph level by level. It starts from a designated source node and systematically explores all the neighbour nodes at the current depth before moving on to nodes at the next depth level.

It uses a first-in-first-out (FIFO) queues to keep track of nodes to be processed. Additionally, BFS can be used to check the connectivity of a graph and to find the connected components in an undirected graph.

b) DFS or Depth-First-search:

It is another graph traversal algorithm, but unlike BFS, DFS explores as far as possible along each branch before backtracking. It starts from a designated source node and explores as deeply as possible along each branch before backtracking. It starts from a designated node.

DFS can be implemented using recursion or an explicit stack data structure. DFS can be used for finding connected components, topological sorting & solving maze problem.

c) Parallel Breadth First Search:

To implement a parallel version of BFS using OpenMP, we can use a shared queue data structure that will hold the vertices to be processed. Each thread will pick a vertex from the queue, process it and add its visited neighbours to the queue. This process will continue until the queue is empty.

In the Implementation, we first create a vector of bools to keep tracks of visited vertices, a queue to hold the vertices to be processed and we push the start the vertex into the queue and mark it as visited. Then we start and openMP parallel region & iterate until the queue is empty.

d) Parallel Depth First Search.

~~Parallel DFS can be Implemented a parallel version of DFS using OpenMP, we can use a stack data structure that will hold the vertices to be processed. Each thread will pick a vertex from the stack, process it and add its unvisited neighbors to the stack. This process will continue until the stack is empty.~~

Parallel DFS works by dividing dividing the graph into smaller subgraph that are explored simultaneously. Each processor or thread is assigned a subgraph to explore and they work independently to explore the subgraph using the standard DFS algorithm.

- Algorithm.

a) Parallel BFS:

```
while queue is not empty:
    #pragma omp parallel for
    for each node in current level of queue:
        processNode (node)
        for each neighbour of node:
            if neighbour is not visited:
                #pragma omp critical
                {
                    mark [neighbour] = true
                    enqueue (queue, neighbour)
                }
    advanceQueue (queue)
```

b) Parallel DFS:

```
while stack is not empty:
    # pragma omp parallel
    {
        current_node = pop (stack)
        # pragma omp critical
        {
            processNode (current_node)
        }
    # pragma omp for
    for each neighbor of current_node:
```

```
if neighbour is not visited:
  # pragma omp critical
  {
      mark [neighbour] = true
      push (stack , neighbour)
   }
```

Conclusion:

We have successfully implemented the parallel breadth first search and depth first search.