HPC 1 DFS

Code

```
#include <iostream>
#include <stack>
#include <vector>
#include <omp.h>
using namespace std;
void dfs_parallel(vector<vector<int>>& graph, int start_node) {
 vector<bool> visited(graph.size(), false);
 stack<int> stack;
 stack.push(start_node);
 while (!stack.empty()) {
 #pragma omp parallel
 {
 int current_node;
 #pragma omp critical
 {
 current_node = stack.top();
 stack.pop();
 }
 if (!visited[current_node]) {
 #pragma omp critical
 {
 visited[current_node] = true;
 cout << "Visiting node: " << current_node << endl;
 }
 // Process neighbors of current_node in parallel
 #pragma omp for
 for (int neighbor : graph[current_node]) {
 if (!visited[neighbor]) {
 #pragma omp critical
 {
 stack.push(neighbor);
 }
 }
 }
 }
 }
 }
}
int main() {
 // Example graph represented as an adjacency list
 vector<vector<int>> graph = {
 {1, 2}, // Node 0
 {0, 2, 3}, // Node 1
 {0, 1, 4}, // Node 2
 {1, 4, 5}, // Node 3
 {2, 3, 6}, // Node 4
 {3}, // Node 5
 {4} // Node 6
 };
 int start_node = 0;
 dfs_parallel(graph, start_node);
 return 0;
}
```

Explanation :

The code performs a parallel Depth-First Search (DFS) traversal on a graph represented as an adjacency list. It starts from a specified starting node and visits each node in the graph, printing a message for each visited node. The code utilizes OpenMP directives to parallelize the processing of nodes and their neighbors, making the traversal more efficient by utilizing multiple threads.

Here's a breakdown of the code:

1. The `dfs_parallel` function takes in the graph (represented as a vector of vectors) and the starting node as parameters.

2. It initializes a vector called `visited` to keep track of visited nodes. Initially, all nodes are marked as not visited.

3. It creates a stack to store nodes for traversal and pushes the starting node onto the stack.

4. The code enters a while loop, which continues until the stack is empty.

5. Inside the parallel region defined by `#pragma omp parallel`, each thread executes the following steps:

   a. The thread retrieves the top node from the stack using `#pragma omp critical` to ensure that only one thread accesses the stack at a time.

   b. If the current node has not been visited, the thread marks it as visited using `#pragma omp critical` to ensure atomicity, prints a message indicating that the node is being visited, and proceeds to process its neighbors in parallel.

   c. Within the parallel region, the loop `#pragma omp for` iterates over the neighbors of the current node.

   d. If a neighbor has not been visited, the thread pushes it onto the stack using `#pragma omp critical` to ensure atomicity.

6. The code continues to execute until all nodes have been visited and the stack becomes empty.

7. In the `main` function, an example graph represented as an adjacency list is defined.

8. The starting node is set to node 0.

9. The `dfs_parallel` function is called, initiating the parallel DFS traversal on the graph.

10. The program outputs messages indicating the visited nodes during the traversal.

11. The program terminates, and the execution completes.

Overall, the code demonstrates how to perform a parallel DFS traversal using OpenMP, making use of multiple threads to explore the graph efficiently.

Oral question:

1. What is DFS - DFS stands for Depth-First Search. It is a popular graph traversal algorithm that explores as far as possible along each branch before backtracking. This algorithm can be used to find the shortest path between two vertices or to traverse a graph in a systematic way. The algorithm starts at the root node and explores as far as possible along each branch before backtracking. The backtracking is done to explore the next branch that has not been explored yet.
   DFS can be implemented using either a recursive or an iterative approach. The recursive approach is simpler to implement but can lead to a stack overflow error for very large graphs. The iterative approach uses a stack to keep track of nodes to be explored and is preferred for larger graphs. DFS can also be used to detect cycles in a graph. If a cycle exists in a graph, the DFS algorithm will eventually reach a node that has already been visited, indicating that a cycle exists.

2. What is Parallel DFS - Parallel Depth-First Search (DFS) is an algorithm that explores the depth of a graph structure to search for nodes. In contrast to a serial DFS algorithm that explores nodes in a sequential manner, parallel DFS algorithms explore nodes in a parallel manner, providing a significant speedup in large graphs.
   Parallel DFS works by dividing the graph into smaller subgraphs that are explored simultaneously.

3. What is OpenMP - OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters. OpenMP is widely used in scientific computing, engineering, and other fields that require high- performance computing. It is supported by most modern compilers and is available on a wide range of platforms, including desktops, servers, and supercomputers