HPC- 1 BFS Code and explanation

```
#include <iostream>
#include <queue>
#include <vector>
#include <omp.h>
using namespace std;
// Function to perform BFS traversal in parallel
void bfs_parallel(vector<vector<int>>& graph, int start_node) {

 queue<int> queue;
 vector<bool> visited(graph.size(), false);

 visited[start_node] = true;
 queue.push(start_node);
 while (!queue.empty()) {
 #pragma omp parallel
 {
 #pragma omp for
 for (int i = 0; i < queue.size(); ++i) {
 int current_node;
 #pragma omp critical
 {
 current_node = queue.front();
 queue.pop();
 }
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
 {
 visited[neighbor] = true;
 queue.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;

 bfs_parallel(graph, start_node);
 return 0;
}
```

Explanation of Code:

1. The code includes necessary libraries for input/output (`iostream`), queues (`queue`), vectors (`vector`), and OpenMP for parallelization (`omp`).

2. The function `bfs_parallel` performs a Breadth-First Search (BFS) traversal on a graph in parallel. It takes a 2D vector `graph` representing the adjacency list of the graph and an `int` `start_node` specifying the starting node for the traversal.

3. Inside the function, a queue is created to store the nodes to be visited. Additionally, a boolean vector `visited` of the same size as the graph is initialized to keep track of visited nodes.

4. The `start_node` is marked as visited and added to the queue.

5. The while loop continues until the queue is empty.

6. The code enters a parallel region using the `#pragma omp parallel` directive.

7. Within the parallel region, the `#pragma omp for` directive is used to distribute the loop iterations among the available threads. The loop iterates over the elements in the queue.

8. Inside the loop, a critical section (`#pragma omp critical`) is used to ensure that only one thread accesses the queue at a time. This is necessary to prevent race conditions when multiple threads try to modify the queue simultaneously.

9. The current node is dequeued from the queue.

10. A message is printed to indicate that the current node is being visited.

11. Another parallel region is created using the `#pragma omp for` directive. This parallel region parallelizes the processing of the neighbors of the current node.

12. The loop iterates over the neighbors of the current node.

13. Inside the loop, a critical section is used to mark the neighbor as visited and add it to the queue if it has not been visited before. Again, the critical section ensures that only one thread modifies the visited array and the queue at a time.

14. The main function initializes the graph as an adjacency list using a vector of vectors. Each inner vector represents the neighbors of a node.

15. The `start_node` is set to 0.

16. The `bfs_parallel` function is called with the graph and the start node as arguments.

17. The program ends with a return statement.

Overall, this code performs a parallel Breadth-First Search traversal on a graph represented as an adjacency list. It uses OpenMP directives to parallelize the processing of nodes and their neighbors. The traversal starts from a specified start node and visits each node in the graph, printing a message for each visited node.

Oral questions

1. What is BFS –
   BFS stands for Breadth-First Search. It is a graph traversal algorithm used to explore all the nodes of a graph or tree systematically, starting from the root node or a specified starting point, and visiting all the neighboring nodes at the current depth level before moving on to the next depth level.

2. Applications of BFS - BFS is commonly used in many applications, such as finding the shortest path between two nodes, solving puzzles, and searching through a tree or graph.

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.

4. What is Parallel BFS?
   Parallel BFS (Breadth-First Search) is an algorithm used to explore all the nodes of a graph or tree systematically in parallel. It is a popular parallel algorithm used for graph traversal in distributed computing, shared-memory systems, and parallel.