Design and implement Parallel Depth First Search on graph based on existing algorithms using OpenMP.

```
#include <iostream>
#include <vector>
#include <stack>
#include <omp.h>

struct Node {
   int data;
   std::vector<Node*> neighbors;
   bool visited;
};

void parallelDFS(Node* start) {
   std::stack<Node*> dfsStack;
   dfsStack.push(start);

   #pragma omp parallel
   {
      while (!dfsStack.empty()) {
         #pragma omp single nowait
         {
            Node* current = dfsStack.top();
            dfsStack.pop();

            if (!current->visited) {
               // Process current node
               std::cout << current->data << " ";
               current->visited = true;

               // Push unvisited neighbors onto the stack
               #pragma omp for
               for (int i = 0; i < current->neighbors.size(); ++i) {
                  Node* neighbor = current->neighbors[i];
                  if (!neighbor->visited) {
                     dfsStack.push(neighbor);
                  }
               }
            }
         }
      }
   }
}

int main() {
   // Create a graph for testing
   Node* node1 = new Node{1, {}, false};
   Node* node2 = new Node{2, {}, false};
   Node* node3 = new Node{3, {}, false};
   Node* node4 = new Node{4, {}, false};
   Node* node5 = new Node{5, {}, false};

   node1->neighbors.push_back(node2);
   node1->neighbors.push_back(node3);
   node2->neighbors.push_back(node4);
   node3->neighbors.push_back(node4);
   node3->neighbors.push_back(node5);

   parallelDFS(node1);

   // Clean up allocated memory
   delete node1;
   delete node2;
   delete node3;
   delete node4;
   delete node5;

   return 0;
}
```