Objašnjenje koda

The code is an implementation of the TSP problem (Travelling Salesman Problem) using the Dynamic Programming algorithm with an OpenMP parallelization.

The TSP problem is defined as finding the shortest path that visits a given set of cities exactly once and returns to the starting city.

The input for the problem is a weighted graph of cities where the edge weight represents the distance between two cities. The output is the shortest path that visits all cities exactly once and returns to the starting city.

The implementation starts by including the required headers, the bits/stdc++.h header provides access to several C++ standard library functions, the chrono header provides functionality for measuring time, and the omp.h header provides OpenMP functionality for parallel programming.

Next, the program defines some global constants and variables. INF is defined as a large number to represent infinity and MAX is defined as the maximum number of cities. n is defined as the number of cities, and g is a 2D array representing the graph, where each element is the weight of the edge between two cities.

The program then defines two global arrays: f and path. f is a 2D array used to store the minimum distance of the TSP tour ending at the cur city and with the bitmask of visited cities. path is a 2D array used to store the next city to visit after reaching the cur city with the bitmask of visited cities.

The tsp function is the main function of the implementation. It takes two arguments, cur and bitmask, representing the current city and the bitmask of visited cities, respectively.

The function checks if the value for the cur city and bitmask has already been computed and stored in the f array. If yes, it returns the value stored in f.

The function then checks if all cities have been visited by comparing the bitmask to (1 << n) - 1, where 1 << n is a bitwise left shift operation that generates a number with n ones, and -1 sets the rightmost bit to 0. If all cities have been visited, the function returns the distance from the cur city to the starting city.

Next, the function declares ans as INF, and uses the #pragma omp parallel directive to parallelize the inner loop that iterates over all cities to find the next city to visit. local_ans and local_path are declared as local variables for each thread to store the minimum distance and next city, respectively.

The #pragma omp for directive is used to indicate that the loop should be executed in parallel by multiple threads. The loop iterates over all cities to find the next city to visit, checking that the next city is not the current city and that the city has not been visited (bit is not set in the bitmask).

If the conditions are satisfied, the function calculates the new cost of visiting the next city as the sum of the distance from the current city to the next city and the minimum distance of the TSP tour ending at the next city with the updated bitmask that has the next city visited.

After all threads have completed, the ans and path are updated with the minimum distance and next city, respectively. The function then returns the minimum distance of the TSP tour.

Finally, the program measures the time it takes to execute the program.

Razlog sporijeg izvršenja za više threadova (u mom slučaju > 7)

The code might be running slower because of overheads introduced by OpenMP, such as creating and synchronizing threads. The TSP problem might not be parallelizable enough to fully benefit from the increased number of threads.

Razlog zašto radi sporije za male matrice

The size of the problem is small, which means that the time spent in synchronization might be larger compared to the time spent in computation.