Name : Manthan Gopal Dhole **TY IT**
Id : 201080020 **Parallel Computing**

Link : Click IT

Experiment 6

Aim: Implement Parallel Traveling Salesman Problem using OpenMP.

Theory :

The Traveling Salesman Problem (TSP) is a well-known optimization problem in which a salesman has to visit a given set of cities exactly once and return to his starting city, while minimizing the total distance traveled. The Parallel TSP (PTSP) is the TSP problem solved in parallel by dividing the search space among different threads.

OpenMP is a popular parallel programming model that allows for easy implementation of parallel algorithms on shared-memory architectures. OpenMP uses a fork-join model, in which a team of threads is created and work is divided among them.

To implement PTSP using OpenMP, we can follow these steps:

Create a list of cities and their coordinates.
Divide the list of cities among the available threads.
Each thread will independently search for the shortest path that visits all the cities assigned to it.
Merge the solutions of all the threads to find the overall shortest path.
The Traveling Salesman Problem (TSP) is a well-known problem in computer science and operations research. It involves finding the shortest possible route that visits a set of cities and returns to the starting city. The problem becomes more complicated when there are multiple salesmen involved, and each salesman has to visit a subset of the cities.

Name : Manthan Gopal Dhole **TY IT**
Id : 201080020 **Parallel Computing**

This problem is known as the Parallel Traveling Salesman Problem (PTSP). In the PTSP, a set of salesmen is given, and each salesman has to visit a subset of the cities. The goal is to find the shortest possible route for each salesman such that all the cities are visited exactly once.

OpenMP is a popular framework for parallel programming in shared memory architectures. It provides a set of directives and libraries that allow developers to parallelize their code easily. In the context of the PTSP, OpenMP can be used to parallelize the computation of the shortest route for each salesman.

The basic idea is to divide the set of salesmen into multiple subsets and assign each subset to a different thread. Each thread will then compute the shortest route for the salesmen in its subset. Once all the threads have finished, the computed routes can be combined to obtain the final solution.

The parallelization of the PTSP using OpenMP can be done in several ways. One approach is to use a shared memory model, where all the threads have access to a shared memory pool. The shared memory can be used to store the distance matrix, the current solution, and any intermediate results. The threads can then synchronize their access to the shared memory to avoid race conditions.

Another approach is to use a distributed memory model, where each thread has its own memory space. In this approach, the distance matrix is divided into smaller sub-matrices, and each thread is assigned a subset of the sub-matrices to compute. Once all the threads have finished, the results can be combined to obtain the final solution.

Code :

```
code = """
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <omp.h>

#define MAX_N 16

int n;
int dist[MAX_N][MAX_N];
```

Name : Manthan Gopal Dhole **TY IT**
Id : 201080020 **Parallel Computing**

```
int visited[MAX_N];
int curr_path[MAX_N];
int min_path[MAX_N];
int min_cost = INT_MAX;

void tsp(int curr_cost, int curr_pos, int level)
{
    if (level == n) {
        // Visited all cities
        curr_cost += dist[curr_pos][0];
        if (curr_cost < min_cost) {
            min_cost = curr_cost;
            #pragma omp critical
            {
                for (int i = 0; i < n; i++) {
                    min_path[i] = curr_path[i];
                }
            }
        }
        return;
    }
    for (int i = 1; i < n; i++) {
        if (!visited[i]) {
            visited[i] = 1;
            curr_path[level] = i;
            tsp(curr_cost + dist[curr_pos][i], i, level + 1);
            visited[i] = 0;
        }
    }
}

int main()
{
    printf("Enter the number of cities: ");
    scanf("%d", &n);

    printf("Enter the distances between the cities:\\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

Name : Manthan Gopal Dhole **TY IT**
Id : 201080020 **Parallel Computing**

```
            scanf("%d", &dist[i][j]);
        }
    }

    visited[0] = 1;
    curr_path[0] = 0;
    tsp(0, 0, 1);

    printf("Path: %d", min_path[0] + 1);
    for (int i = 1; i < n; i++) {
        printf("->%d", min_path[i] + 1);
    }
    printf("->%d\\n", min_path[0] + 1);
    printf("Minimum Cost/Minimum weight Hamiltonian Cycle: %d\\n",
min_cost);

    return 0;
}



"""
```

Output :

```
!./Travelling_salesman

Enter the number of cities: 3
Enter the distances between the cities:
3
5
2
55
6
2
66
2
4
Path: 1->3->2->1
Minimum Cost/Minimum weight Hamiltonian Cycle: 59
```

Name : Manthan Gopal Dhole **TY IT**
Id : 201080020 **Parallel Computing**

Conclusion:

In summary, OpenMP can be used to parallelize the PTSP by dividing the set of salesmen into multiple subsets and assigning each subset to a different thread. The threads can then compute the shortest route for the salesmen in their subset and synchronize their results to obtain the final solution. The choice of shared or distributed memory model depends on the specific requirements of the problem and the available resources.