COL 380	March 1, 2016

Lab 1

Instructor: Subodh Sharma	Due: March 8, 23:55 hrs

1 Parallelizing Traveling Salesperson Problem (TSP) using OpenMP

Consider the famous problem of traveling salesperson, TSP. Given a list of cities and the cost of traveling between each pair of cities, the problem is that the salesperson must visit each city once returning to her hometown and she must do this with the least possible cost. The route starts from her hometown.

1.1 Tree-based Search

TSP is known to be a NP-Hard problem that results in a computational explosion with a time complexity of O(n!). In this assignment, you have to develop a non-recursive parallel implementation of TSP that is based on tree-search. The tree's root indicates the hometown and the leaves indicate complete tours (Hamiltonian circuit) ending in hometown. The internal nodes of the tree are partial tours.

An approach to solving TSP is to maintain a best cost of the tour traversed so far; any thread whose partial tour cost exceeds the best cost can be abandoned. The tour-tree can either be explored in a DFS, in BFS or in combination. Explain clearly in few sentences the strategy that you choose and the rationale behind it. Implement the chosen strategy. In the parallel implementation, one has to also think on how to divide partial tours to threads for further exploration. The only requirement of this assignment is that assigning the workload to threads should be performed **statically** (i.e., you will have to explicitly code the work assignment strategy). Explain the design of load-balancing in the lab-report.

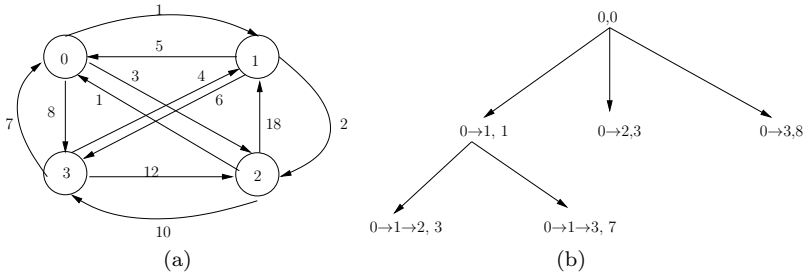

Figure 1: TSP for 4 cities and associated partial tree of tours

The input to the problem is given in the form of a matrix. An element $a[i][j]$ indicates the cost of traversing the route between the cities i and j. Pseudo-code of an iterative serial implementation is shown below:

```
Push_copy(stack,tour); // tour that visit only hometown
while (!Empty(stack)){
  curr_tour = Pop (stack);
  if(City_count(curr_tour) == n) {
```

```
    if(Best_tour(curr_tour))
       Update_best_tour(curr_tour);
  } else {
    for (nbr = n-1; nbr >=1; nbr--)
      if (Feasible(curr_tour, nbr)) {
        Add_city(curr_tour, nbr);
        Push_copy (stack, curr_tour);
        Remove_last_city(curr_tour);
      }
   }
}
```

The definitions of the functions used in the pseudo-code are explained in the source file provided with this assignment.

1.2 Statement of work

- Implement the function parallel_tree_search() and associated functions to complete the parallelization of TSP.
- Prepare a lab report (as .txt file) by clearly explaining your design decisions, parallelization strategy, load-balancing strategy. Make sure that explanations are bulleted and are not long paragraphs. Keep the lab report within 200-400 words. To compile your programs use the following command:

  ```
  gcc -fopenmp -O3 -Wall -o tsp tsp.c
  ```

  Make sure that while debugging option `-O3` is replaced by `-g`. In order to debug use DDD, GDB, or IDE-based debuggers.

Turn in the source code file and the report. The break-down on grading is as follows:

- Correct execution of the program **40 Marks**
- Report clarity **10 Marks**

1.3 Files Provided

- An incomplete tsp.c
- A script file to generate the inputs, generate_di_graph.py