ECE1747H Assignment 1 Report

Haocheng Wei (1008498261)
Yuanze Yang (1009209269)
Yizhou Shen (1004536308)

System Infomation

For this assignment, we run our codes on ug-lab machines. The system attributes are shown in table 1.

Thread per Core	1
Cores per Socket	8
Total Sockets	1
Total Online CPUs	8

Table 1: CPU information

Improvements Walkthrough

In our proposed parallel version, we create a vector of queues that provides one queue per thread. This design enables each thread to explore multiple partial paths, as in Figure 1. In comparison, in the original sequential code, only one queue is provided for partial paths, causing latency for path expansion.

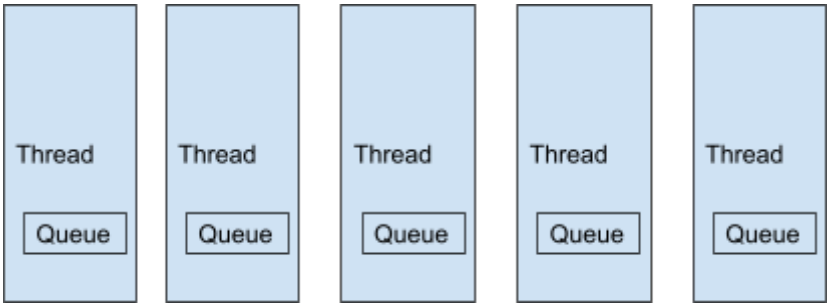

Figure 1: Overall parallel structure

The vector of queues also avoids the overhead by synchronizing a one-queue-only parallel structure, in which one thread only expands one partial path, as in figure 2.

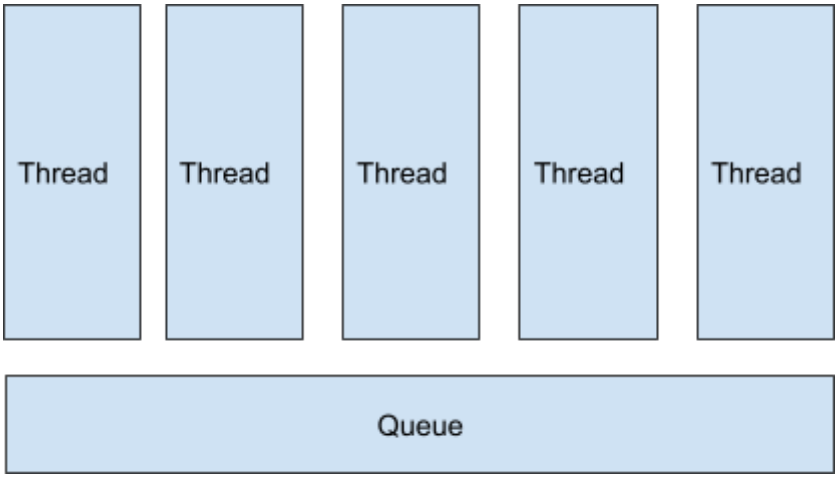

Figure 2: One queue structure for parallelization

Performance Analysis

Table 2 shows the execution time of sequential and parallel solutions.

Number of Cities	Sequential Solution (ms)	Parallel Solution (ms)		
		2 Threads	4 Threads	8 Threads
3	0	0	0	0
6	0	0	1	1
9	3	2	1	1
12	17	13	10	7
14	160	116	72	63
16	4745	3443	2054	1676
18	29347	20458	8878	5081
20	173791	134886	81233	57067

Table 2: Performance Analysis of TSP problem under different solutions.

In our parallel version, we separately executed the code with the parameter MAX_THREAD = 2, 4, and 8. From the table, we can find out that:

1. When the number of cities is under 10, the performance of sequential code and parallel ones is basically the same. Especially when thread count is over 4 and the number of cities is relatively small, multi-threading even shows its disadvantage.
2. The advantage of multi-threading shows itself when the number of cities rises. under the scale of 20 cities, the solution time is reduced by 68% using a maximum of 8 threads. Given the number of 18 cities, this improvement stands out as almost ⅛.
3. Multi-threading can significantly speed up the process of solution. The more thread is used, the faster the execution time. However, the thread-time relation is not linear.

In the aggregate, when the problem scale is not that large (<10 cities), the overhead from synchronization and creating/joining threads causes performance degeneration; as the problem scales up, the multi-threading solution will exhibit its inherent concurrent merit.