Assignment 1: TSP

Charvi Choksi | 1006952317
Shwetha Padmanabhan | 1007541242
Nirmit Zinzuwadia | 1002434074

Improvements Added

- Instead of a single queue, we create **4** different queues when the number of cities exceeds 4. Each queue is handled by a different thread. The reason why we choose 4 threads instead of other numbers is due to the hardware specs of the ug-lab machines. From the CPU specs, we get the following data:
  - **Thread(s) per core**: 1
  - **Core(s) per socket**: 4
  - **Socket(s)**: 1
  - We have 4 CPU sockets, each CPU has 1 core, each with a single thread. Hence, max thread count: 4 CPU x 1 cores x 1 threads per core = 4
- We utilize a single variable `shortestPath` to store the optimal solution. This variable is protected by mutex since it will get updated by multiple threads.

Visual Representation of the Solution

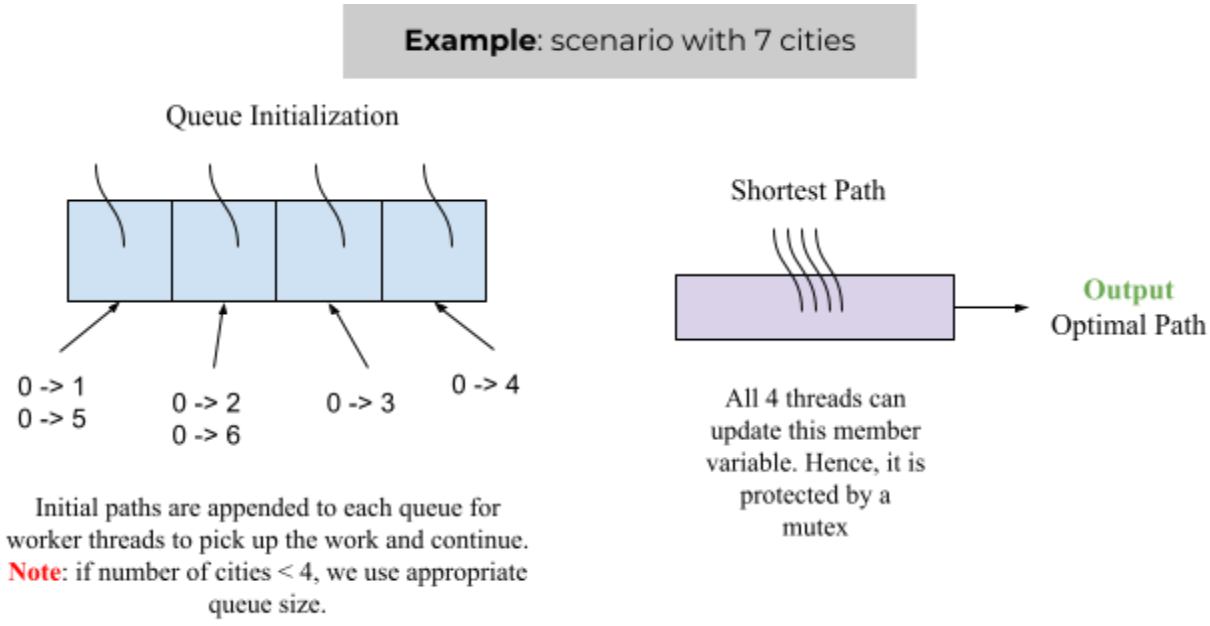

Figure 1: Visual representation of how the code is working. The above example looks at 7 cities but this idea can be generalized.

Analysis

As part of the experiment, we also analyzed the performance impact with a greater number of threads. *Table 1* shows the performance numbers. As the number of threads increases, we don't necessarily see a greater improvement. This is because the cost of hyperthreading outweighs the performance improvements through greater threads. We see a slight improvement when the number of cities increases with 8 threads. This is because, with hyperthreading, the optimal solution is found early on, hence the cost of calculating further paths is decremented significantly.

Performance Table

Number of Cities	Sequential Solution (ms)	Parallel Solution (ms)		
		4 Threads	6 Threads	8 Threads
3	0	0	0	0
9	1	0	1	1
11	24	9	11	11
13	173	69	86	79
15	1011	498	514	630
17	10474	2031	2640	2439
18	52070	16083	16842	12985
20	302202	148746	197764	137306

Table 1: Performance between Sequential and Parallel Execution