Travelling Salesman Problem

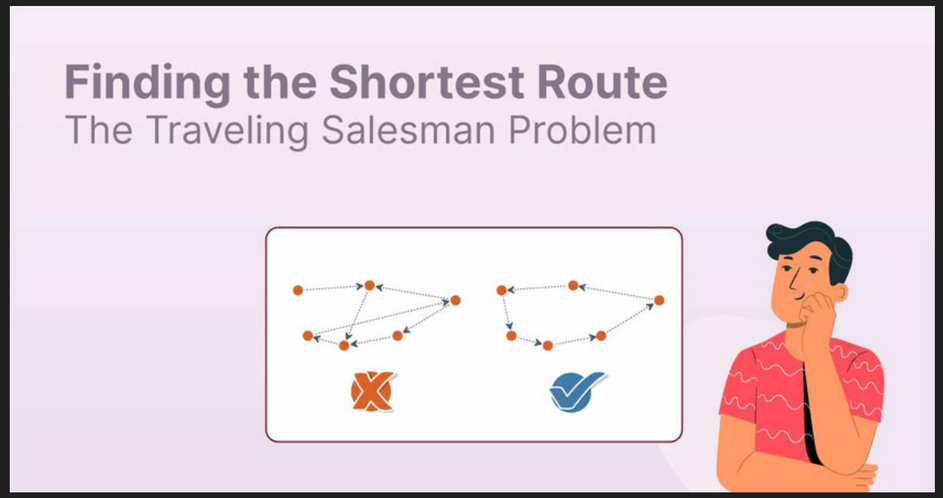
Parallel Implementation

Shriram V | P V Rishi Dev

What is the problem ?

- **The Traveling Salesman Problem (TSP)** is a well-known problem in the realm of optimization, where the goal is to determine the most efficient route a salesperson should take to visit a set of cities and return to the starting city.

- The challenge of finding the shortest yet most efficient route for a person to take given a list of specific destinations along with the cost of travelling between each pair of destinations. The problem is an NP-Hard problem that is, no polynomial-time solution exists for this problem.



Problem Objective

- **Objective:** Implement and analyze parallel solutions for the Traveling Salesman Problem (TSP).
  - Develop parallelized algorithms using OpenMP, MPI and Hybrid (OpenMP + MPI).
  - Assess and compare the performance, speedup, and efficiency of each parallelization approach.
  - Contribute insights into the effectiveness of parallel computing for optimizing TSP solutions.
- **Outcome:** Enhance understanding of parallel algorithms and their impact on solving complex optimization problems, specifically the TSP.

Solution Overview

Parallel Solutions:

- **OpenMP:** Shared memory processing, using multiple threads to optimize TSP calculations on a single machine.

- **MPI:** Message Passing Interface, facilitating communication and collaboration between multiple processors for TSP parallelization.

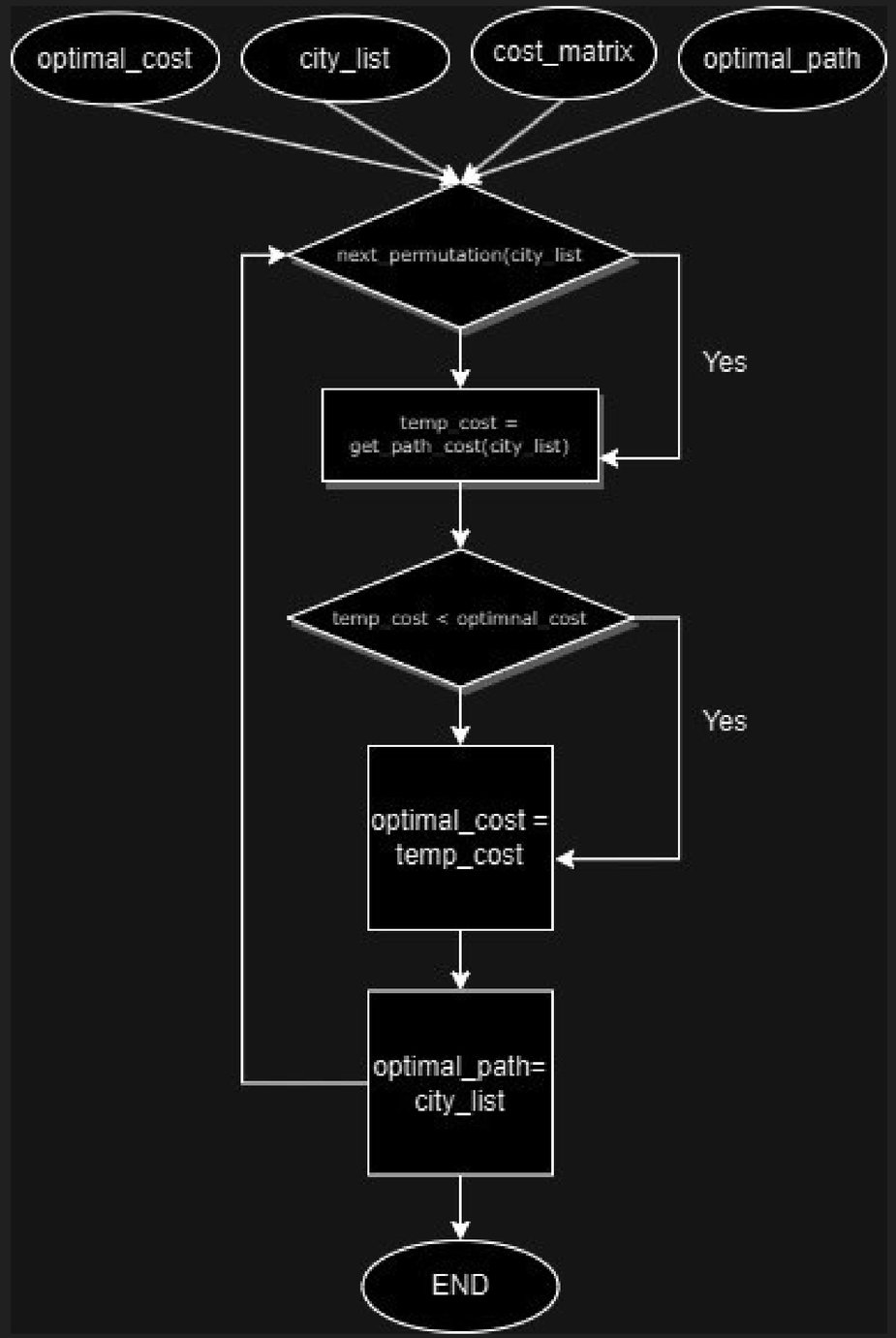
- **Hybrid:** Combining OpenMP and MPI to leverage both shared memory and distributed processing for enhanced efficiency

Solution Overview (contd.)

- **Non-Parallel Solution:**
  - **Brute Force Algorithm:** Sequential approach involving exhaustive permutation search to find the optimal TSP solution.

Explore diverse parallelization techniques and compare their performance against a non-parallel brute-force approach for the Traveling Salesman Problem.

Block Diagram



Solution (OpenMP)

- Parallelization done by dividing (N-1)! permutations among available threads.
- Execution time decreases with increase in # of threads.
- The difference between the execution time for two consecutive cities is large because it depends on N!.
- Speedup increases with increase in the number of threads, while the trend remains similar across all N's.

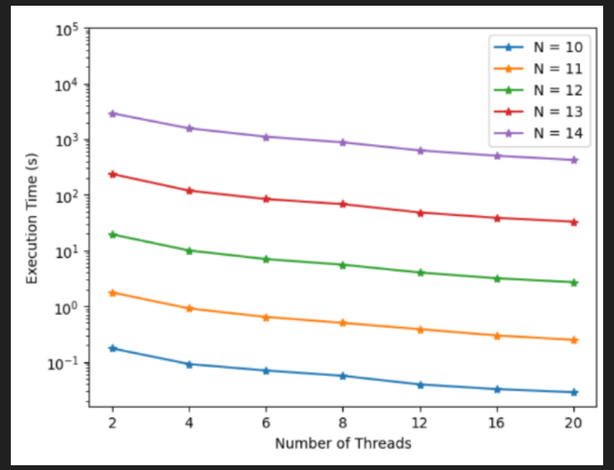


Figure: Variation of execution times with number of OpenMP threads for different problem sizes (N=Number of cities)

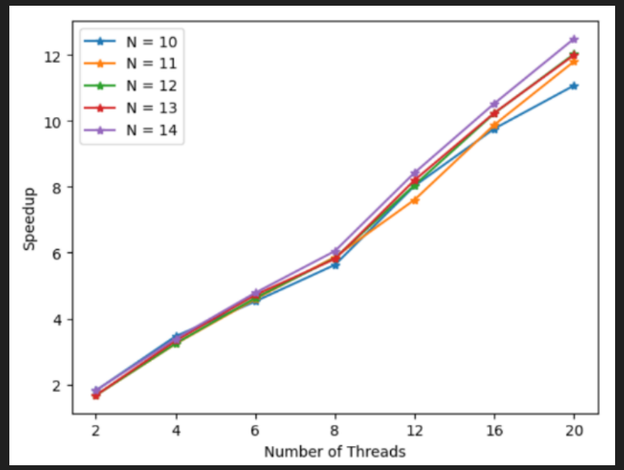


Figure: Variation of speedup achieved with num ber of OpenMP threads for different problem sizes (N=Number of cities)

Solution (MPI)

- Parallelization done by dividing (N-1)! permutations among available PEs.
- Execution time decreases with increase in # of PEs.
- The difference between the execution time for two consecutive cities is large because it depends on N!.
- Speedup increases with increase in the number of threads, while the trend remains similar across all N's.

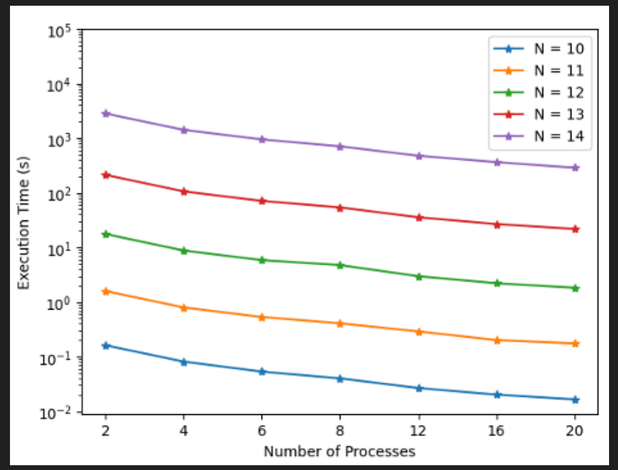


Figure: Variation of execution times with number of MPI processes for different problem sizes (N=Number of cities)

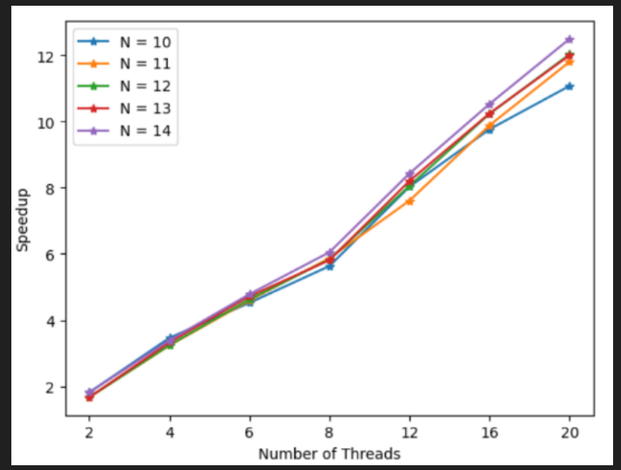


Figure: Variation of speedup achieved with number of MPI processes for different problem sizes (N=Number of cities)

Comparitive Analysis (OpenMP & MPI)

- Speedups achieved through MPI are greater than those achieved through OpenMP
- Using 20 PEs(in MPI) results in speedups of 18.
- Using 20 threads(inOpenMP) results in speedups of merely 12.
- Efficiency for MPI code is always greater than its OpenMP counterpart
  - Potentially due to extra over-heads of spawning & maintaining OpenMP threads compared to the inter-process communications in MPI.

		No. of Threads (p)				
		2	4	8	16	20
N	10	0.907	0.865	0.704	0.610	0.554
	11	0.834	0.808	0.734	0.618	0.590
	12	0.834	0.810	0.729	0.639	0.601
	13	0.834	0.831	0.726	0.640	0.600
	14	0.909	0.848	0.750	0.658	0.618

Table: Efficiency( )of the OpenMPi mplementation for various N & p(no. of OpenMP threads)

		No. of PEs (p)				
		2	4	8	16	20
N	10	0.975	0.970	0.984	0.976	0.953
	11	0.922	0.920	0.898	0.910	0.838
	12	0.918	0.922	0.851	0.916	0.886
	13	0.927	0.929	0.913	0.921	0.904
	14	0.934	0.931	0.931	0.908	0.918

Table: Efficiency ( ) of theMPI implementation for various N & p(no. of MPI Processes)

Solution (Hybrid)

Parallelization Paradigm		Execution Time (s)
OpenMP (20 threads)		32.934
MPI (20 PEs)		21.866
Hybrid	2 PEs × 10 threads	124.736
	4 PEs × 5 threads	62.219
	5 PEs × 4 threads	50.492
	10 PEs × 2 threads	25.822

- Spawn multiple OpenMP threads(num_threads) in each of the multiple MPI processes(num_PEs).
- Permutations are divided equally among the *num_PEs X num_threads* parallel elements.
- With same total number of parallel elements, increasing the number of MPI processes reduces the execution time.
  - Larger overheads of maintaining OpenMP threads compared to the MPI communication overhead.
- Using lesser number of PEs(with same total parallel elements) proves to be worse even compared to OpenMP
  - Higher costs of maintaining threads & their synchronization across only a small number of PEs compared to single processor.
  - Once the number of PEs becomes large enough, thread maintenance costs are over come by MPI communication.

References

Travelling Salesman Problem: Parallel Implementations & Analysis
Amey Gohil, Manan Tayal, Tezan Sahu, Vyankatesh Sawalpurkar

Thanks

Shriram V | P V Rishi Dev