Travelling Salesman Problem

Meghana G S
Dept. Information Technology(B. Tech)
National Institute of Technology
Surathkal, Karnataka, India
meghanags.201it136@nitk.edu.in

Shalini C E
Dept. Information Technology(B. Tech)
National Institute of Technology
Surathkal, Karnataka, India
shalinice.201it256@nitk.edu.in

Yuvraj Singh Jadon
Dept. Information Technology(B. Tech)
National Institute of Technology
Surathkal, Karnataka, India
yuvrajsinghjadon.201it268@nitk.edu.in

***Abstract*—The travelling salesman problem is problem where the task is finding the shortest path between given points and places that has to be visited. In the problem points are the cities a salesperson has to visit. The goal is to keep the distance travelled by the person should be minimum and each city should be visited at least once. Understanding The Traveling Salesman Problem can be solutions to the challenge for the logistics and supply chain industry. With multiple vehicles, more cities and multiple sales professionals, TSP gets tougher to crack. It is easier to solve the problem in theory because you have to find the shortest route for every trip within a city. But It gets harder as the number of cities increases. Hence this paper is dedicated towards the parallelisation of the TSP solution to make it even more efficient. We have included three different parallel solutions and detailed comparisons between them with graphs are provided in the report.**

I. Introduction

Travelling salesperson problem: As the name suggests it's the problem about travelling a particular number of cities and coming back to the starting point and the distance covered should be minimum. Suppose the cities are x1 x2..... xn where cost cij denotes the cost of travelling from city xi to xj. The travelling salesperson's problem is finding a route starting and ending at x1 that will take all cities with the minimum cost.

Our goal here is to first study the brute force algorithm and the Dynamic approach and then seek to parallelisation and see if can achieve more efficiency. And also provide a detailed comparison of the different parallel approaches that we are going to use.

II. Serial Implementation of the solution

A path through every vertex exactly once is the same as ordering the vertex in some way. Thus, to calculate the minimum cost of travelling through every vertex exactly once, we can brute force every single one of the N! permutations of the numbers from 1 to N.

A. Approach

All the possible paths or the arrangements of the cities are considered, compared to find the most optimal path i.e, the minimum cost path to travel all the cities.

B. Time Complexity

There are N! permutations to go through and the cost of each path is calculated in O(N), thus this algorithm takes O(N * N!) time to output the exact answer.

III. Motivation to Parallelize the solution

The brute force solution for the travelling salesman problem traces every possible path from a city to another city i.e, all the permutations and find the path whose cost is minimum. This solution takes exponential time to find the path when it's executed sequentially, thus in this project we aim to parallelize the brute force solution.

In this project, we use 3 different paradigms of parallelizing the solution:

- OpenMP (Shared memory Processing)
- MPI (Message Passsing Interface)
- CUDA (for GPUs)

We also compare and analyze the computation time taken by the different methods of parallelization and the sequential execution in this paper.

IV. OpenMP(Shared Memory Processing)

A. Approach

To parallelize the solution of travelling salesaman problem in OpenMP, we considered all the possible arrangements of the cities and they were divided among the number of threads equally. If the number of permutations are not divisible by the number of threads, then the first r(remaining arrangements) threads are given extra one arrangement. To maintain the continuity, the initial arrangements for each thread is calculated based on the thread ID and the index will be given to the thread, then the consecutive arrangements will be figured out inside the threads. Each thread will calculate the optimal path among the arrangements it executes and then all the thread optimum paths will be considered and the most optimal path will be the one which is the answer.

B. Analysis

Figures 1 and 2 show the analysis of the computation time taken by the OpenMP program and the speedup achieved for different number of cities and the threads.

- The computation time decreases as the number of threads increase.
- The difference between the execution time of consecutive cities is large as it depends on N!.
- We can infer from the graph that the speedup increases as the number of threads increase, while the trend remains similar for all N's.

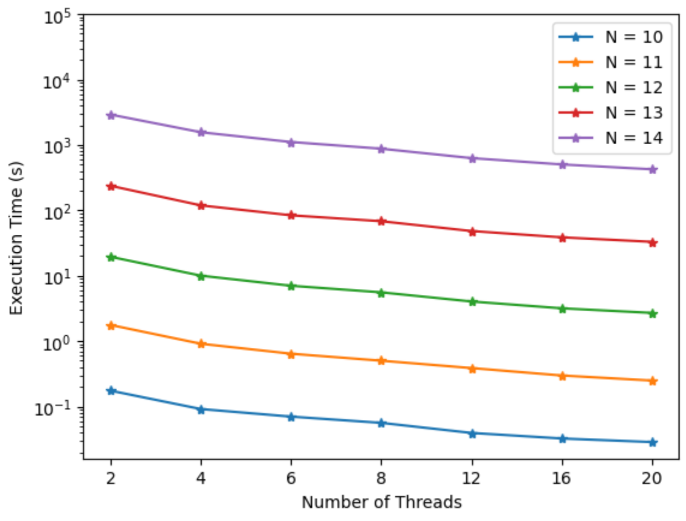

Fig. 1. Variation of execution times with number of OpenMP threads for different problem sizes (N=Number of cities)

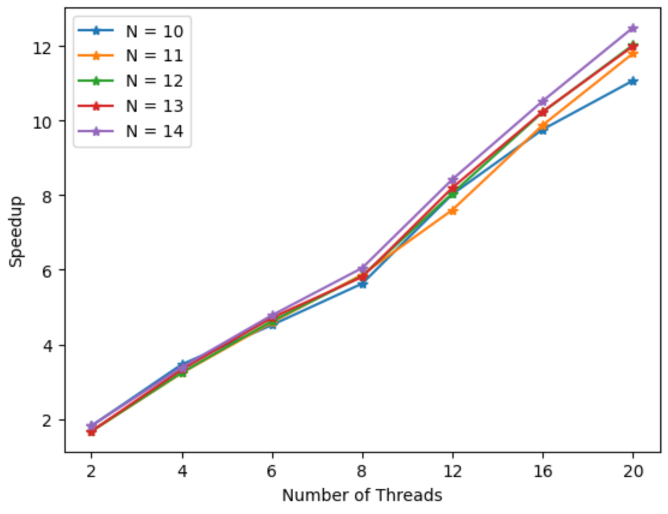

Fig. 2. Variation of speedup achieved with number of OpenMP threads for different problem sizes (N=Number of cities)

V. MPI(Message Passing Interface)

A. Approach

To parallelize the solution of travelling salesman problem in Message Passing Interface(MPI), all the arrangements or permutations are divided among the available Parallel Environments (PEs). The case when the number of permutations are not divisible by the number of available PEs is taken care by dividing the remaining arrangements among the first PEs possible. The starting index and the ending index of the permutations are provided for each of the PEs, thus the consecutive permutations are taken care inside the PEs. Each of the PEs find the optimal path among all the possible paths provided to them. Once all the PEs finish execution, they are synchronized and the most optimal path among all values will be calculated in the Master PE.

B. Analysis

Figures 1 and 2 show the analysis of the computation time taken by the MPI program and the speedup achieved for different number of cities and the Parallel Environments (PEs).

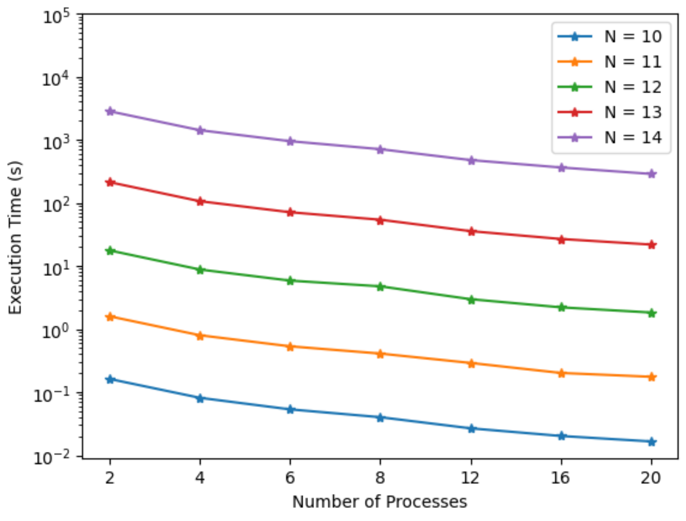

Fig. 3. Variation of execution times with number of MPI processes for different problem sizes (N=Number of cities)

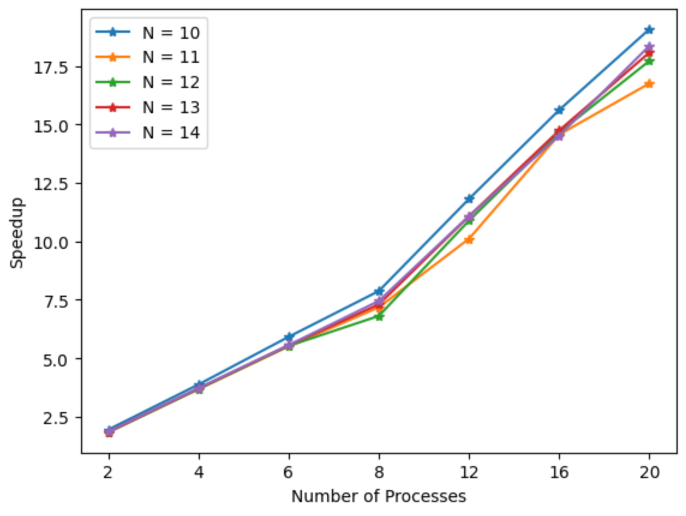

Fig. 4. Variation of speedup achieved with number of MPI processes for different problem sizes (N=Number of cities)

- The computation time decreases as the number of PEs increase.
- The difference between the execution time of consecutive cities is large as it depends on N!.
- We can infer from the graph that the speedup increases as the number of PEs increase, while the trend remains similar for all N's.

VI. Comparative study of OpenMP and MPI

On the basis of the timing analysis of the OpenMP and MPI implementations ,we can conclude that the speedups that are achieved through the Message Passing (i.e., MPI) are greater or speeder than those achieved through the Shared Memory (i.e., OpenMP). This is the apparent from the fact that using 20 PEs (in MPI) can results in speedups of 18, while using the 20 threads (in OpenMP) results in the speedups of merely 12.

		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

Fig. 5. Efficiency of the OpenMP implementation for various N (problem size) 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

Fig. 6. Efficiency of the MPI implementation for various N (problem size) p (no. of MPI Processes))

This is also an evident by comparing the efficiencies of MPI and OpenMP implementations shown in Tables III IV. Clearly it states , the efficiency for MPI code is always greater or larger than its OpenMP counterpart. This is potentially due to the extra overheads of spawning athe nd maintaining OpenMP threads compared to the inter-process communications in the MPI. We can also calculate the Karp-Flatt Metric, or experimentally can determined serial fraction e(N, p) of parallel computation for both OpenMP and MPI implementations.

		No. of Threads (p)				
		2	4	8	16	20
N	10	0.103	0.052	0.060	0.043	0.042
	11	0.199	0.079	0.052	0.041	0.037
	12	0.199	0.078	0.053	0.038	0.035
	13	0.198	0.068	0.054	0.037	0.035
	14	0.100	0.060	0.048	0.035	0.033

Fig. 7. e(N, p) of the OpenMP implementation for various N (problem size) p (no. of OpenMP threads)

- For a given N, e(N, p) usually decreases or remains constant with increase in p, indicating that the brute force TSP algorithm is rather embarrassingly parallel..

		No. of PEs (p)				
		2	4	8	16	20
N	10	0.025	0.010	0.002	0.002	0.003
	11	0.085	0.029	0.016	0.007	0.010
	12	0.089	0.028	0.025	0.006	0.007
	13	0.078	0.026	0.014	0.006	0.006
	14	0.070	0.025	0.011	0.007	0.005

Fig. 8. e(N, p) of the MPI implementation for various N (problem size) p (no. of MPI Processes)

- Comparing the corresponding e(N, p) values for OpenMP MPI, highlights that OpenMP code contains higher fraction of serial component, leading to lower speedups.

VII. CUDA(NVIDIA GPUs)

A. Approach

For parallelizing of the brute force approach for Travelling salesman problem using CUDA, we can leverage both: blocks threads. For the our analysis, we used 50 blocks, each with the 1024 threads. And We divide the permutations of all the cities among the threads in all the blocks. Each thread in a block which calculates the minimum cost path from the assigned permutations in the block. After we synchronizing the each threads in each separate block, one thread (with Thread ID = 0) computes the optimal path cost for the block, and stores it in a shared global DS that can be accessed by both CPU and GPU. Once the GPU which finishes its execution, the host will calculates the optimal path of the TSP by iterating over the minimum cost path of each block stored in the previously mentioned dS.

B. Analysis

The below table summarizes the time taken for execution and speedup achieved by leveraging the power of NVIDIA GPU.

N	Time (s)		Speedup
	Serial Code	CUDA Code	
8	0.004	0.017	0.264x
9	0.065	0.042	1.537x
10	0.316	0.020	15.725x
11	2.9416	0.057	51.278x
12	32.440	0.048	673.260x
13	395.165	0.086	4557.187x
14	5289.317	0.822	6427.022x
15	74060 (∼ 20 hr) §	9.753	7590x †
16	1110900 (∼ 308 hr) §	151.387	7340x †
17	∼ 4936 hr §	1867.08	9520x †

The below Graph shows the Execution times speedups achieved using CUDA for various values of N.

Some observations that made from the above analyze include:

- As N increases, it is shows that the GPU provides immense benefits through parallelization, as we get speedups of great magnitudes.

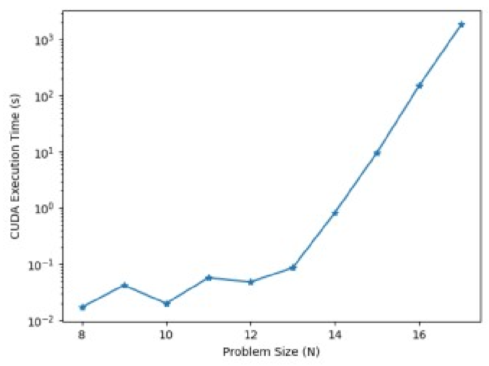

Fig. 9. : Plot of CUDA execution time w.r.t problem size (N))

Above fig shows the variation of CUDA program execution time with increase inthe problem size N. We were also restricted by the §Approximated using O(N!) time of serial algorithm †Estimated using approximated serial time actual time of the CUDA time maximum capacity of C++ primitive datatypes to keep our analysis only upto N = 17.

- For large N (15), it was always infeasible to run the serial code due to the estimated total time based on the O(N!) that is the nature of the algorithm.
- For N 13, even though the speedups are huge and large , the actual CUDA execution time increases or speed ups exponentially because N! becomes overwhelmingly large or huge to be parallelized across 50 × 1024 threads.
- Following the abouve trend, if N = 18, the serial execution could take aapproximately 83900 hrs. Now, even if we extrapolate the speedups to 10000x, the CUDA program code will still require 8+ hrs to execute, which is unacceptably the large.
- This analysis clearly illustrates or states that imperative need to choose an efficient algorithm to solve a problem in CUDA instead of trying to parallelize an inefficient algorithm.

Conclusion

In this paper, we executed the brute force solution for the Travelling Salesman Problem(TSP) in sequential as well as parallel environments and compared the computation time among different environments. We implemented the solution in OpenMP, MPI and CUDA programming and made the detailed time analysis of each of the paradigm. We also conducted the in depth analysis between OpenMP and MPI computation time. This project gives the better idea of how an algorithm performs in different environments.

References

[1] We referred this link for the OpenMP parallelization https://tildesites.bowdoin.edu/ ltoma/teaching/cs3225-GIS/fall17/Lectures/openmp.html: :text=OpenMP

[2] For the MPI parallel exections the reffered the link to get more info about it https://ubccr.freshdesk.com/support/solutions/articles/13000010161-mpi-and-parallel-computing.

[3] For CUDA paralle execution program we referred the given link for the reference https://www.infoworld.com/article/3299703/what-is-cuda-parallel-programming-for-gpus.html.

[4] For the serial Execution of the TSP problem ,we referred the gfg for the info about the problem https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/