Solving Traveling Salesman Problem using GPU

Artem Komarov

February 17, 2023

1 Summary

The project aims to solve the NP-hard Traveling Salesman Problem using both CPU and GPU to compare the performances. The brute force algorithm will be used for better parallelization and to show the performance difference between CPU and GPU. Another aim of the project was creating a nice project structure with files-folders structure, Google testing, CMake, and Git.

2 Background

The Traveling Salesman Problem (TSP) is one of the most famous optimization problems in computer science. It asks the question: what is the shortest possible route that a salesman can take to visit a given set of cities and return to the starting point? Despite its simple definition, the TSP is known to be an NP-hard problem, meaning that it is computationally intractable to find the optimal solution for large problem sizes. This has made the TSP a popular benchmark problem for algorithmic research, as well as a practical challenge.

Parallel computing is a powerful tool for solving computationally intensive problems such as the TSP. Using GPU parallelization, we can offer significant speedup in execution time, making it possible to tackle larger problem sizes. GPUs, in particular, have been shown to be very effective for solving TSP problems due to their high number of cores and efficient memory access.

In this project, we aim to solve the TSP using both CPU and GPU implementations, with the goal of comparing their performances. We decided to use a brute force algorithm for solving the TSP, as it provides a simple approach that can be easily parallelized. We use domain decomposition to distribute the workload, meaning that we split the search space equally between the threads. Also we focused on comparing the execution time of our CPU and GPU implementations, so will see how better GPU performance is. (Or worse? We will answer this question below)

3 Challenge

Initially, our challenge was to identify a suitable TSP solution that could be effectively implemented on both CPU and GPU. We faced with additional obstacles when implementing the path distribution process that would examine each thread (from which path to start the loop?). Locating the currentPath memory posed another challenge, because the pathSize is a variable. Collecting the results together was also a challenge, but since the results were only collected at the end, the workload was manageable. So regarding to the workload, the memory was global, but used only by one thread, and therefore the cache worked well. We also tried using shared memory, which showed additional speed up. The computation complexity was $O(n^n)$, and the communication complexity was $O(n)$. Divergences were also not a significant concern because communication occurred only at the end of the programm - collecting the result.

4 Approach

In this project, the brute force algorithm was used for better parallelization. Brute force algorithms means checking all the possible paths and choosing the best one. Domain decomposition was used to equally split the data between threads. (we do not care about the complexity of the solution because

Threads number	TSP solving time	TSP solving time with shared memory usage
64	524.9	499.5
128	266.9	253.1
256	139.8	133.4
512	79.9	74.5
1024	56.4	55.8
CPU	374.6	-

Table 1: CPU and GPU performance

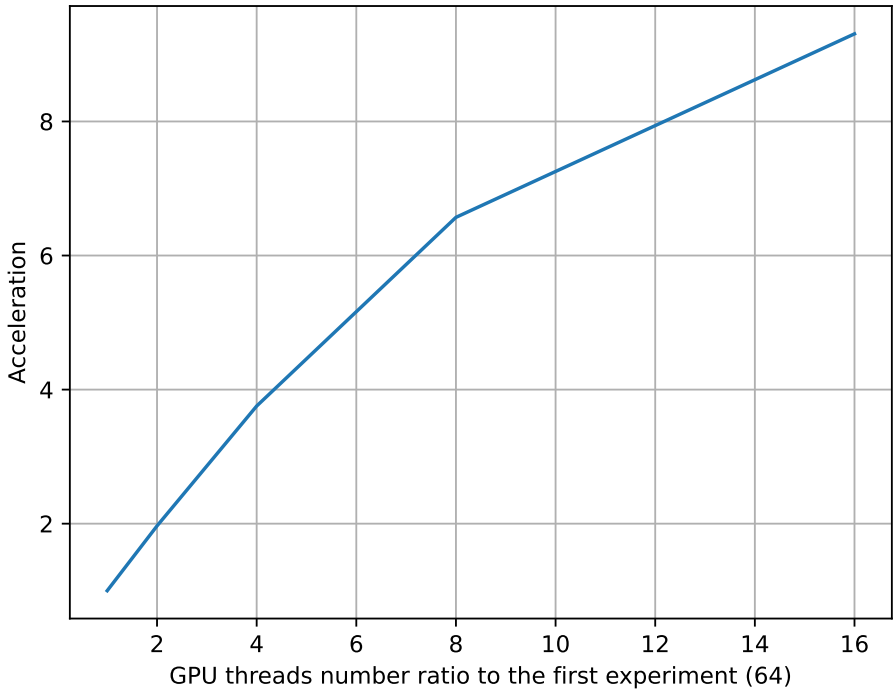

Figure 1: GPU acceleration

it is more important for us to compare performance and do good parallelization). The implementation has been done using CUDA and C++. Different thread numbers were tested for GPU to find the optimal thread number. The process of optimization involved trying out more complex TSP solutions but they were difficult to rewrite on the GPU, and the parallelization was not that good. Another step of optimization was shared memory usage.

5 Results

Main experiments were conducted on a problem with 11 towns. The distances table you can find at the Appendix 1. The optimal path length turned out to be 133. Different thread numbers were tested for GPU, and the solving times are shown in the Table 1. We built the acceleration graph, which you can find on the Figure 1. The code you can find on the github: https://github.com/ArtKomarov/TSP_GPU.

6 Conclusion

The project successfully solved the NP-hard TSP problem using GPU and provided a comparison of performances with CPU. We achieved a good acceleration. Also we managed to create a nice project structure with the different build possibilities for running on CPU, GPU, and for testing.

7 References

1. https://cmake.org/
2. https://developer.nvidia.com/
3. https://docs.nvidia.com
4. https://en.wikipedia.org/wiki/Travelling_salesman_problem
5. https://github.com/ArtKomarov/TSP_GPU
6. https://people.sc.fsu.edu/ jburkardt/datasets/tsp/tsp.html

8 Appendix 1

0	8	50	31	12	48	36	2	5	39	10
8	0	38	9	33	37	22	6	4	14	32
50	38	0	11	55	1	23	46	41	17	52
31	9	11	0	44	13	16	19	25	18	42
12	33	55	44	0	54	53	30	28	45	7
48	37	1	13	54	0	26	47	40	24	51
36	22	23	16	53	26	0	29	35	34	49
2	6	46	19	30	47	29	0	3	27	15
5	4	41	25	28	40	35	3	0	20	21
39	14	17	18	45	24	34	27	20	0	43
10	32	52	42	7	51	49	15	21	43	0

Table 2: Distances table between towns