# Optimised GPU implementation of Travelling Salesmen Problem

Nihala Karapoola
Computer Science Engineering
National Institute of Technology Karnataka
India
Email: nihalakp@gmail.com

Giffin Suresh
Computer Science Engineering
National Institute of Technology Karnataka
India
Email: giffinsuresh97@gmail.com

***Abstract*—Graphics Processing Units(GPU) are throughput oriented processors that is specifically used to accelerate parallel computation tasks. This paper serves to utilizes the above concept to optimize the famous traveling salesman problem. Several methods of optimization are discussed with special focus on the Ant-Colony Optimization, a heuristic algorithm that is developed observing the behavior of common ants. These methods are used to achieve a significant speed-up. The algorithm also takes advantage of the common optimization methods such as shared memory access and memory coalescing. On an average, compared to an efficient GPU implementation, it approximately takes 32 CPUs with 8 cores each (256 cores total) to give the same performance.**

## I. Problem Statement

The famous TSP problem can be stated as Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. In other words we need to find the shortest Hamiltonian Cycle considering the cities to be vertices and the routes to be edges.

## II. Objectives

The objective is to implement the problem statement parallely using GPU. Owing to the fact that the deterministic approach does not give a polynomial time, we resort to heuristic methods. We propose to compare the serial and parallel approaches by their execution times and prove that parallel approach provides a significant speed up.

## III. Work Distribution

1. Serial Implementation of TSP: Nihala

2. Naive Parellel implementation : Giffin

3 Recording the runtimes and plotting graph for serial implementations:Giffin

4. Recording the runtimes and plotting graph for naive parellel implementations:Nihala

5. Optimizations-Memory Coalescing and Shared memory,Ant Colony : Giffin and Nihala

6.Mid Progress report and Discussion : Giffin and Nihala

7.Include more Optimizations based on the results obtained : Combined effort

8.Coming with an algorithm that improves state of the art : Combined effort

## IV. Project Timeline

Given below is the timeline for the project:

1.*October 8th*: 1.Serial Implementation of TSP 2.Parellel Implementation of TSP

2.*October 11th*: 1.Recording runtimes and plotting graph for Serial Implementations 2.Recording runtimes and plotting graph for parallel Implementations

3.*October 20th*: Including Optimizations 1.Memory Coalescing and Shared Memory 2.Ant Colony

4.*October 23rd*: Mid-Progress Report and Discussion

5.*November 5th*: Include more optimizations

6.*November 11th*: Improving state of the art

7.*Novemeber 13th*: Final project Submission

## V. Serial ACO Algorithm

To begin the algorithm, an ant is placed on each node of the graph. At each iteraion of the algorithmm every ant chooses a new node to go to. Each edge has two weights. The first weight is proportional to the distance between two cities. The second weight is fundamental to the ACO scheme, it is the pheromone weight. The ant picks an edge based on a random roll relative to these two weights. These ant choosing iterations are repeated until the ants have completed a full tour of the graph. Upon all the ants completing a full tour of the graph the ant with the shortest tour is picked and a global decay is applied to all the edges pheromone weights. The algorithm then takes the best ants tour and adds to the relevant edges pheromone weight accumulator. The purpose of this step is to emphasis this edge in future ant walks. The algorithm is repeated until the ants begin to converge on a result.

Serial ACO

1: procedure ACO(g) .Returns best path in g

*2: ants .Array of ants*

*3: while not converged do*

*4: for all ants  ant do*

*5: ant.tour*

*6: end for*

*7: decay all pheromone trails*

*8: f ind best ant trail*

*9: increment all edges on best trail*

*10: end while*

*11: end procedure*

## VI. Parallel ACO Algorithm

The first task necessary in parallelizing ACO is the partitioning of the graph. Potential partitioning schemes include: round robin, clusters distribution, random distribution among other methods. After partitioning the graph, the ACO algorithm is run on each node.

During each iteration an ant attempts to make a tour. However, it will inevitably find that it needs to visit a node that is absent from the processor responsible for it. At this point, the ant is sent over to the proper processor. Once it is received there, it joins the rest of the ants on that processor in the queue waiting its turn to process. After being sent around to every processor at least once, visiting all of the nodes in the graph, the ant has completed its tour. The processors then do a reduce amongst themselves to see which ant(s) had the best tour. A reduction is then done over all processors that sums the total number of ants that have shortest tours. Every processor then expects to receive that many ants minus the number of shortest tour ants that it currently has. This is the sync step, now all processors have a copy of the best tour ants. Each processor then updates the pheromone weights using the best ants tours. This can be somewhat tricky to implement with all of the coordination that must go on in order to keep the processes from having unexpected behavior.

Parallel ACO *1: procedure ACO(g) .Returns best path in g*

*2: ants .Array of ants*

*3: while not converged do*

*4: while every ant has not visited my nodes do*

*5: for all ants  ant do*

*6: ant.tour*

*7: end for*

*8: receive new ants*

*9: add new ants to ants queue*

*10: end while*

*11: perform reduction to determine smallest tour*

*12: count = reduction sum of total smallest tour ants*

*13: if I have a smallest tour ant then*

*14: send ant to all*

*15: end if*

*16: receive ( count  my smallest tour count ) ants*

*17: decay all pheromone trails*

*18: find best ant trail*

*19: increment all edges on best trail*

*20: end while*

*21: end procedure*

*22: procedure Ant Tour(ant) .Parallel Ant Tour*

*23: while all nodes not visited do*

*24: Determine edge*

*25: if edge is not on processor then*

*26: send ant to processor with edge*

*27: break*

*28: end if*

*29: Take edge*

*30: end while*

*31: end procedure*

## VII. Comparison

Our algorithm is written such that it takes advantage of n cores, from 1 to k, where k is the maximum number of cores available to the system. Large problem sizes will have difficulty fitting on a single cores worth of memory. Our algorithm is intended to require less memory per processor as the number of processors increases, so the more processors you have the bigger the problem size you can handle even if each processor has a relatively small amount of memory. From the graphs above, we can see that the parallelized version was successful at least as far as outperforming its serial competitor; the parallel version being roughly 7 times faster on the largest test case that was run on Kratos.

## VIII. Conclusion

TSP is one of the most popular optimization problems, currently being worked on. This optimization will lead to a significant speed boost in applications such as in maps to find the shortest route, optimizing tourist destination visits, drilling of printed circuit boards, Crystallography and so on. Hence achieving this will ensure a remarkable contribution to the field of Parallel Computing.

The Ant Colony Optimization problem sets out to solve an NP-hard problem, giving little hope to a well scaling serial solution. We set out to adapt this algorithm to the parallel programming domain by splitting up the graph problem into process sized chunks. By spliting up the graph we necessarily complicate the problem slightly by requiring a communication scheme that allows us to maintain an ant taking a complete tour across the graph by moving between the processes. It is also required that we can sync the best trail across all of the partial graphs.

## REFERENCES


[1] Akihiro Uchida, Yasuaki Ito and Koji Nakano, An Efficient GPU Implementation of Ant Colony Optimization for the Travelling Salesman Problem, in Third International Conference on Networking and Computing, 2012.

[2] Molly A. ONeil, Dan Tamir and Martin Burtscer, A Parallel GPU Version of the Travelling Salesman Problem.

[3] Kamil Rocki and Reiji Suda, Accelerating 2-opt and 3-opt Local Search Using GPU in the Travelling Salesman Problem.

[4] Th Van Luong, Nouredine Melab, El-Ghazali Talbi. GPU Computing for Parallel Local Search Metaheuristics. IEEE Transactions on Computers, Institute of Electrical and Electronics En- gineers, 2013, 62 (1), pp.173-185.

[5] Olfa Bali, Walid Elloumi, Pavel Kromer and Adel M. Alemi, GPU Particle Swarm Optimization Applied to Travelling Salesman Problem in 2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip