Efficient Parallel GPU Implementation of ACO-RFD Hybrid Algorithm to Solve Travelling Salesman Problem

Ankit Shrivastava, Krishna Manglani
Department of Computational and Data Sciences
Indian Institute of Science, Bangalore, India
ankitvaibhava@gmail.com, krish2100@gmail.com

***Abstract*—The Travelling Salesman Problem(TSP) is one of the most paradigmatic NP-problem.Over years many heuristic algorithms have been proposed to solve TSP.Ant colony optimization(ACO) and river flow dynamics(RFD), two related swarm intelligence methods have been used to solve TSP in past.In this paper we solve TSP using a parallel implementation of hybridized ACO-RFD on GPU. We have used different strategies to hybrid ACO with RFD and compared the solution quality.We have implemented these algorithms on GPU using both task parallel and data parallel approach and compared the speedup with serial and existing parallel versions.**

I. Introduction

Ant colony optimization (ACO) algorithm model the behaviour of real ants.Real ants uses the pheromone deposited by other ants to follow the path.This behaviour has been used while designing ant algorithm where artificial ant constructs a solution for a given problem by carrying out random walk on graph.The probability of traversing a certain edge depends on the pheromone level deposited on that edge.

River formation dynamics (RFD) is another swarm method. RFD is based on copying how river path is formed in nature due to erosion and sedimentation. These feature has been used in RFD algorithm. Each node is assigned some value called altitude.Artificial drop traversing a path in graph chooses its edge based on maximum negative gradient of the edge.Here gradient is difference of altitudes of nodes divided by edgelength.Thus, RFD can be considered as gradient version of ACO.

The paper by Rabanal, Pablo and Rodrguez [1] mentions that the standard ACO algorithm tends to stuck in local cycle.Whereas in RFD this local cycle is not possible due to its gradient based approach.Also RFD has other advantages over ACO such as fast reinforcement of new shortcuts and a localized method to punish blind alleys. Another paper by Rabanal, Pablo and Rodrguez [2] says that although RFD has advantages over ACO, but the application of RFD to TSP is less natural than application of ACO to TSP.In general solutions constructed by RFD are better, though ACO obtains acceptable solutions faster.

Hence need of hybrid algorithm arises. This hybrid algorithm tends to obtain best characteristics of ACO and RFD.By using hybrid algorithm our focus is on getting better result than ACO but with very little compromise in speed.Instead of using ants and drops to traverse the node a new entity ant-drop that contains all the attributes of ants as well as all the attributes of drops is created.

Since in hybrid algorithm there will be large amount of ant-drop entity which randomly chooses the starting nodes and independently traverse the graph without any interaction among each other it provides a great opportunity to implement this code on GPU using NVIDIA CUDA.In GPU a thread or thread block can act as an ant-drop entity.

The rest of the report is organised as follows: In Section-2, Related Work of our hybrid execution is summarised.Section-3 We talks about our hybrid implementation methodology wherein we describe all the steps in details. In Section-4, experiment setup and results are presented along with analysis.We finally conclude in Section-5.

II. Related Work

For years many have proposed improved version of ACO.Dorigo, M. and Gambardella, L.m. [3] discusses the ACO to solve TSP. In this they have shown that ACO have shown some favourable results with other algorithms like genetic algorithms, evolutionary programming, and simulated annealing. They have also concluded that constructive methods to generate good starting solutions can be considered as good strategy.

Rabanal, Pablo [2] discuss the implementation of hybrid ACO-RFD algorithm to solve TSP problem.They tries to extract best feature of both ACO and RFD.Thus they are able to get reasonable solution as compared to ACO with reasonable time as compared to RFD.

Uchida, Akihiro [4] discusses the parallel implementation of ACO on GPU. They have tried to parallelize tour construction stage and pheromone update stage using data parallel approach.They were able to conclude that with their new parallel

algorithm compared to its sequential counterpart executes up to 82x faster while preserving the quality and 8.5 times faster than other existing parallel implementation on GPU.

III. Methodology

In this section, we provide a complete description of the algorithm that we employed for the ACO-RFD hybrid implementation on CUDA. Algorithm 1. explains basic scheme of ACO-RFD hybrid algorithm as explained by Rabanal, Pablo and Rodrguez [2]. This scheme consists of following phases:

- initializePheromonesAndNodes
- initializeAntDrops
- moveAntDrops
- depositPheromones
- evaporatePheromones
- erodePaths
- depositSediments

Algorithm 1 Hybrid Algorithm Scheme

```
initializePheromonesAndNodes()
initializeAntDrops()
while not endingCondition() do
    moveAntDrops()
    depositPheromones()
    evaporatePheromones()
    erodePaths()
    depositSediments()
    analyzePaths()
end while
```

We have implemented moveant() phase algorithm using three different strategies on CUDA which we will be discussing one by one.

Following are the three strategies

- hybrid-1 (HB1) using task parallel
- hybrid-2 (HB2) using task parallel
- hybrid-2 (HB2) using data parallel

Before starting any execution in CUDA first we execute two phases in CPU i.e. initializePheromonesAndNodes where we initialize pheromone level at each edge by INI_TPH and altitude of each node by INIT_ALT.Then we initialize each ant-drop by putting them in one of the nodes of the graph.Once initialization is done complete data is loaded into GPU.Thus the pheromone level, altitude of each node and cost matrix are stored in global memory of GPU to which each thread has access.

A. Hybrid-1 (HB1) using task parallel

First weightage of w_aco and w_rfd is fixed before execution. Here in GPU each thread work as an ant-drop. Each thread has access to their separate respective array such as entity-path, edge-probability stored in global memory. Then the thread in moveantdrop phase chooses the next path in parallel using following equation

$$P(i,j) = w_{aco}P_{ant}(i,j) + w_{rfd}P_{drop}(i,j)$$

Algorithm 2. explains the basic scheme of HB1 moveantdrop() phase

Algorithm 2 Move Agents Kernel (Task Parallel HB-1)

```
for each AntDrop in parallel do
    Set Visited Array to zero
    Set CurrentCity = id mod num_cities
    Set City = 0
    while City < num_cities do
        i = CurrentCity
        Schedule(City) = CurrentCity
        visited(i) = 1
        for each unvisited city j do
            P_ant(i,j) = (τ_ij^α)(η_ij^β) / Σ_j(τ_ij^α)(η_ij^β)
            P_drop(i,j) = decreasingGradient_ij / Σ_j decreasingGradient_ij
            P(i,j) = w_aco P_ant(i,j) + w_rfd P_drop(i,j)
        end for
        CurrentCity = arg max_j P(i,j)
        City++
    end while
    Set pathlength(id) = find_cost(Schedule)
```

B. Hybrid-2 (HB2) using task parallel

Here similar to previous algorithm each thread act as separate entity and execute moveant phase in parallel. The only difference is that instead of calculation the probability by thread to choose next path, it first randomly decides the nature whether it will act as an ant or as a drop before starting moveant() phase as per following equation.

if (nature = 0)
$P(i,j) = P_{ant}(i,j)$
else
$P(i,j) = P_{drop}(i,j)$

C. Hybrid-2 (HB2) using data parallel

Here similar to previous algorithm each entity first randomly decides the nature whether it will act as an ant or as a drop before starting moveant(). But major difference in this strategy is that instead of each thread acting as separate entity here each thread block act as separate ant-drop entity with fixed number of threads for each thread block.This further help in parallelizing moveant() phase.Also another benefit here is that earlier each entity had to acess global memory again and again during a single moveant() phase this time it will have to access its own shared memory. Also while computing the pathlength parallel reduction takes place

which further parallelize the algorithm. Algorithm 4. explains the basic scheme of HB2-data parallel moveantdrop() phase.

Algorithm 3 Move Agents Kernel (Data Parallel HB-2)

for each thread in threadBlock in parallel **do**
sharedVariables: nature,CurrentCity,nextCity[N],prob[N]
Set Visited Array to zero collectively
Master thread sets:
CurrentCity = id mod num_cities
nature
City = 0
while City $<$ num_cities **do**
i = CurrentCity
Master Thread Sets:
Schedule(City) = CurrentCity
visited(i) = 1
prob(threadId) = 0
temp = 0
for each unvisited and associated city j **do**
if (nature = 0)
temp = $P_{ant}(i,j)$
else
temp = $P_{drop}(i,j)$

if (prob(threadId) $<$ temp)
prob(threadId) = temp
nextCity[threadId] = j
end for
__syncthreads()
Compute $\arg\max_j P(i,j)$ using parallel reduction and store in master thread
Master thread sets CurrentCity = nextCity(Master ThreadId)
City++
__syncthreads()
end while
Find cost of schedule in parallel and store in Master thread
Master Thread Sets Set pathlength(id) = schedule cost

Once all threads complete traversing the graph we parallely update pheromone and altitude data in rest of the phases such as depositpheromone,evaporatepheromene, as explained by Dorigo, M. [3] and depositesediment, erode path as explained by Rabanal, Pablo [5]

The above phases are repeated for next iterations and the complete algorithm is executed for specified number of iterations.

IV. Experiments and Results

A. Experiment Setup

The following results were obtained by running on Tesla K40m cluster with driver version 7.5. It contains 2880 CUDA cores, 745 mhz as GPU max clock rate.

B. Results

The main motive behind parallelizing the hybrid algorithm was to obtain best possible optimal tour in fastest possible way available.The table clearly shows that for 1000 iterations we are able to get better results using hybrid algorithm over simple ant colony algorithm individually.

TABLE I
Cost results comparison for 1000 iterations

City	ACO	HB1	HB2	Optimal path
berlin52.tsp	8093	8009.2	8172	7544.4
kroa100.tsp	9282	9238	9244.03	7944.8
a280.tsp	3343	3149.7	2830	2586.8
pcb442.tsp	58758.5	58953	58530.4	50783.5

It is clear from the figures Fig.1. and in Fig.2. that we are able to achieve speedups in all the cases.For smaller graphs the computation is low so we get low speedup. As the size increases the computation plays major role hence higher speedup is achieved

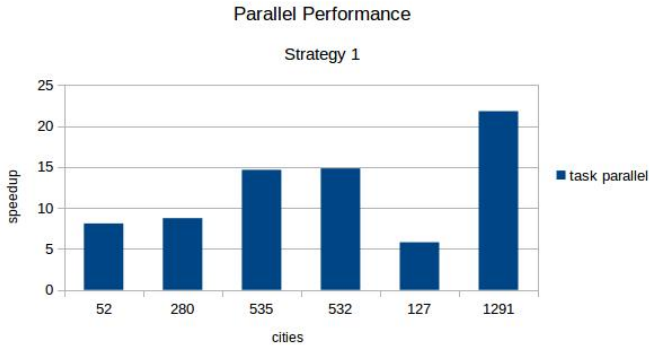

Fig. 1. Speed-ups for HB1

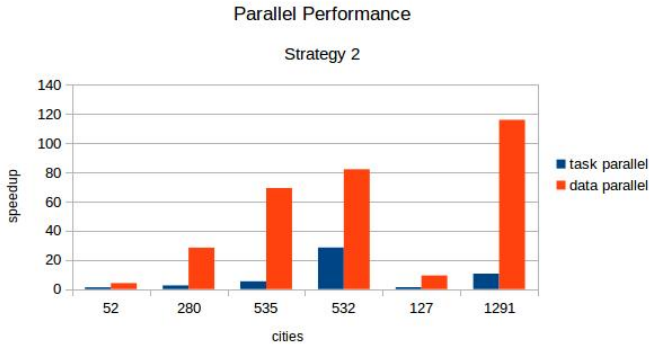

Fig. 2. Speed-ups for HB2

In case of task parallel algorithm all variables are stored in global memory and all thread tries to access their part

of global memory during each phase of algorithm. Because of this the global memory access time plays a major role. While in data parallel while finding next edge the pheromone array is accessed by many threads in such a way that access locations are consecutive. This leads to memory coalescing effect which increases memory throughput and thus increase speedup. Also the computation of pathlengths is done with the help of shared memory and parallel reduction which improves the speed of kernel. Hence we get more speed-up in data parallel than in task parallel.

We have also compared the results of ACO-RFD hybrid CUDA implementation with that of results by Molly A. ONeil [6] in Fig.3 anf Fig.4 .

The multiprocessor occupancy analysis is done to choose the optimal thread block size. The multiprocessor occupancy is ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU. The occupancy is determined by amount of shared memory and registers used by each threadblock. Maximizing occupancy can help to cover memory latency during global loads and increase overall throughput of GPU. The GPU occupancy calculator can assist in choosing thread block size based on shared memory and register requirements. With the kernels provided in methodology, the maximum occupancy with HB-1 strategy was found to be 63 % where as with HB-2 strategy, it was found to be 100 % with the best choice of thread block size of 256.

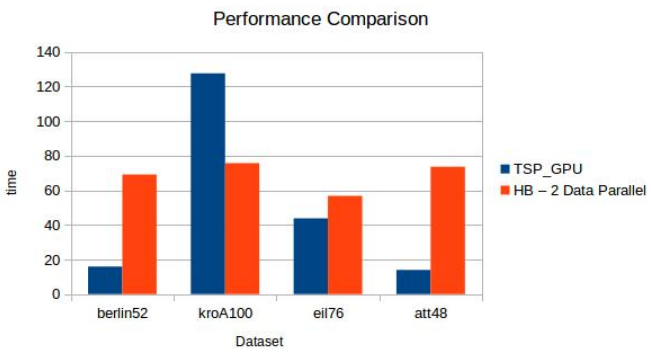

Fig. 3. Performance comparison of ACO-RFD hybrid with standard paper by A. ONeil [6]

V. Conclusions

River flow dynamics is relatively new algorithm with very little work done.We believe that such effective use of RFD in ACO-RFD hybrid algorithm to solve TSP very efficiently have been proposed for first time by As Rabanal, Pablo and Rodrguez [2].Hence we believe that we are the first to parallelize this hybrid algorithm to best of our knowledge.

There is a lot of scope in improving RFD algorithm which inturn help in improvement of ACO-RFD hybrid algorithm. In

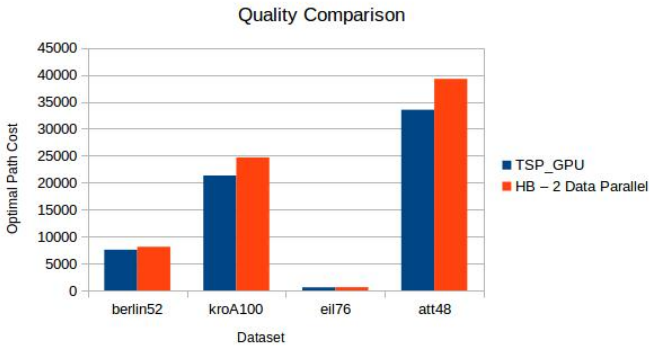

Fig. 4. Quality comparison of ACO-RFD hybrid with standard paper by A. ONeil [6]

our CUDA based hybrid algorithm all phases of algorithm was executed by GPU hence if openMP-CUDA or MPI-CUDA hybridization is used the complete computation-dominated phase can be done by GPU and memory-access-dominated phase can be done by CPUs.Further parallelization of remaining phases of algorithm is also possible.

References

[1] P. Rabanal, I. Rodrguez, and F. Rubio, "Solving dynamic tsp by using river formation dynamics," *2008 Fourth International Conference on Natural Computation*, 2008.

[2] ——, "An aco-rfd hybrid method to solve np-complete problems," *Frontiers of Computer Science Front. Comput. Sci.*, vol. 7, no. 5, p. 729744, 2013.

[3] "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation IEEE Trans. Evol. Computat.*, vol. 1, no. 1.

[4] A. Uchida, Y. Ito, and K. Nakano, "An efficient gpu implementation of ant colony optimization for the traveling salesman problem," *2012 Third International Conference on Networking and Computing*, 2012.

[5] P. Rabanal, I. Rodrguez, and F. Rubio, "Using river formation dynamics to design heuristic algorithms," *Lecture Notes in Computer Science Unconventional Computation*, p. 163177.

[6] M. A. Oneil, D. Tamir, and M. Burtscher, "A parallel gpu version of the traveling salesman problem," in *2011 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2011, pp. 348–353.