Parallel Ant Colony Optimization for Travelling Salesman Problem

Ajinkya Potdar (108738610)
Siddhesh Shirsat (108805949)
Viviktesh Agwan (108943735)

Problem Definition:
Ant colony algorithms are known to have ability to find good quality solutions for some of the problems like Travelling Salesman Problem in a heuristic manner. Traditionally, sequential ACO algorithm sends the ants along the graph in a sequential manner with each ant updating the pheromone model. However, the computational time of these methods is compromised when the current instance of the problem has a high dimension and/or is hard to solve. Thus a lot of study has been done in this field to reduce computation time and improve the solution quality of ACO algorithms by using parallel computing. Parallel computing has become attractive during the last decade as an instrument to improve the efficiency of population-based methods.

In our project, we will be exploring different approaches of parallelizing ACO approach by concurrently sending 'y' number of ants and then updating the pheromone model. This process will be repeated for a certain number of sequential batches (x). We will be studying this problem to (i) reduce the execution time to solve TSP, (ii) enable to increase the size of the problem.

Ant Colony Optimization:
Ants use anon-centralized, population based mechanism to efficiently forage for food in nature. Their mechanism is capable of finding an optimal route in dynamic and varied environments without any external guidance or control and with no central coordination.

It achieves a collective performance which could not normally be achieved by an individual acting alone. ACO is a collection of probabilistic optimization techniques inspired from ants' behavior in nature, for classic combinatorial problems. ACO makes use of artificial ant colonies and a set of software agents called artificial ants. Problem is transformed into the problem of finding the best path on a weighted graph by the artificial ants. The construction process is incremental, stochastic and biased by a set of a pheromone model, whose values are modified at run-time by ants.

Review of the Literature:
Over the past few years, significant work has been done on parallel implementation ofACO, especially for the powerful massively parallel architecture of the GPU.

The strategies applied to the GPU were based on the intrinsic data-parallelism provided by the vertex processor and the fragment processor. The initial experiments compared a grid implementation with 32 workstations equipped with CPUs Intel Pentium IV at 2.4GHz against one workstation with a GPU NVIDIA GeForce 6600 GT. Both strategies were equivalent in terms of the quality of the obtained solutions. The later experiments compared both the GPU parallel strategies, showing that the strategy applied to the fragment processor performed about 35% faster than the strategy applied to the vertex processor.

Then there is parallel implementation of MMAS which uses multiple colonies, where each colony is associated with a work-group and ants are associated with work-items within each work-group. The CPU initializes the pheromone trails, parameters, and also controls the iteration process, while the GPU is responsible for running the main steps of the algorithm: solution construction, choice of the best solution, and pheromone evaporation and updating. Six instances from the Travelling Salesman Problem library (TSPLIB), containing up to 400 cities, were solved using a workstation with a CPU AMD Athlon X2 3600+ running at 1.9GHz and a GPU NVIDIA GeForce GTX 8800 at1.35GHz with 128 processing elements. The parallel GPU version was 2 to 32 times faster than the sequential version, whereas the solutions quality of the parallel version outperformed all the three MMAS serial versions.

There is also a parallel ACO algorithm with a pattern search procedure to solve continuous functions with bound constraints. The parallel method was compared with a serial CPU implementation. Each work-item is responsible for evaluating the solution's costs and constraints, constructing solutions and improving them via. a local search procedure, while the CPU controls the initialization process, pheromone evaporation and updating, the sorting of the generated solutions, and the updating of the probability vectors. The experiments were executed on a workstation equipped with a CPU Intel Xeon E5420 at 2.5GHz and a GPU NVIDIA GeForce GTX 280 at 1296MHz and 240processing elements. The computational experiments showed acceleration values between128 and almost 404 in the parallel GPU implementation. On the other hand, both the parallel and serial versions obtained satisfactory results. However, regarding the solution quality under a time limit of one second, the parallel version outperformed the sequential one in most of the test problems.

In a MATLAB implementation of parallel MMAS has also been researched upon, the authors proposed an algorithm implementation which arranges the data into large scale matrices, taking advantage of the fact that the integration of MATLAB with the Jacket accelerator handles matrices on the GPU more naturally and efficiently than it could do with other data types. Therefore, auxiliary matrices were created, besides the usual matrices (τandη) in a standard ACO algorithm. Instances from the TSPLIB were solved using a workstation with a CPU Intel i7 at 3.3GHz and GPU NVIDIA Tesla C1060 at 1.3GHz and 240 processing elements. Given a fixed number of iterations, the experimental evaluation showed that the CPU and GPU implementations obtained similar results, yet the parallel GPU version being much faster than the CPU. The speedup values had been growing with the number of TSP nodes, but when the number of nodes reached 439 the growth could not be sustained and slowed down drastically due to the frequent data-transfer operations between the CPU and GPU.

The ACO algorithm proposed was implemented on a GPU device, where the parallelism strategies follow a strategy where ants work in parallel to obtain a solution to the problem. The hardware architecture was not available but the computational experiments showed that the GPU version was 15 times faster than its corresponding CPU implementation.

The Quadratic Assignment Problem (QAP) was solved by the parallel ACO based algorithm. Besides the initialization process, all the algorithm steps are performed on the GPU, and all data (pheromone matrix, set of solutions, etc.) are located in the global memory of the GPU. Therefore, no data was needed to be transferred between the CPU and GPU, only the best-so-far solution which checks if the termination condition is satisfied. The authors focus on a parallelism strategy for the 2-opt local search procedure since, from previews experiments, this was the most costly step. The experiments were done in a workstation.

Sequential Algorithm:

ACO (x, y)

x = batch count

y = size of each batch

```
1. Place each ant in a randomly chosen city
2. For x batches of ants of size y
3. For each ant in the batch:
   a. Choose next city
   b. if more cities to visit:
      a. go to 3a
   c. else
      a. return to the initial city
4. update pheromone level using the tour cost of each ant
5. Print best tour
```

Parallel Algorithm:

Parallel_ACO (x, y)

x = batch count

y = size of each batch

```
1. Place each ant in a randomly chosen city
2. pheromone model = uniform
3. For x batches of ants of size y
4. Parallel For each ant in the batch:
   a. Choose next city based on the pheromone model thus constructed
   d. if more cities to visit:
      a. go to 3a
   e. else
      a. return to the initial city
5. Parallel update pheromone model using the tour cost of each ant
6. Sync.Print best tour
```

Expected Results:

Implement the sequential ACO.

Implement the parallel ACO.

The variation in the accuracy of the result (proximity to the best known solution) against the time required to converge to the solution for a given number of cities. These variations will be noted for the sequential as well as the parallel approach.

Plan for one month:

In one month, we plan to implement the serial and the parallel algorithm and test it for a graph with modest number of nodes (say <100). We then plan to plot the variation in the time required and the proximity towards the correct answer as given by each of the algorithms.

In the later time period (after one month), we plan to stress test both the algorithms with significantly large number of nodes (> 100 and increasing till approximately 500). We then plan to measure the variations in the time required and the proximity towards the correct answer as given by each of the algorithms.

We also plan to test the scalability and work optimality by introducing more number of cores and noting the change in the time of execution and the output.

References:

[1] Parallel Ant Colony Optimization for the Traveling Salesman Problem, Max Manfrin, Mauro Birattari, Thomas Stutzle, and Marco Dorigo
[2] Marco Dorigo and Thomas Stutzle. Ant Colony Optimization. The MIT Press, 2004.
[3] Thomas Stutzle. Parallelization strategies for ant colony optimization. In Proc. Of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature, pages 722–731. Springer-Verlag, 1998
[4] Strategies for Parallel Ant Colony Optimization on Graphics Processing Units,Jaqueline S. Angelo, Douglas A. Augusto and Helio J. C. Barbosa