Project 6 : Multi-Colony Ant Optimization for TSP

Due date : May 4, 2012

1 Background

For your second CUDA project, you will implement a variant of the Ant Colony Optimization (ACO) method using multiple ant colonies instead of just one. ACO is a probabilistic optimization technique which is based off of ant behavior and swarm intelligence. It was first proposed by Marco Dorigo in is 1992 PhD thesis for finding optimal paths in a graph, but has since been extended to many other problem domains including protein folding, vehicle routing, and network routing. Unlike some genetic algorithms and simulated annealing, ACO can update in realtime to changes in the system (ie. graph), which makes it useful for realtime routing problems.

In the natural world, ants wander randomly and lay down pheremone trails as they search the world. The more pheremone on a particular trail, the more ants which are attracted to it. To allow ants to explore random trails, the pheremone left by an ant evaporates over time. ACO emulates this behavior by simulating the ant exploration and pheremone evaporation.

2 Requirements

You must implement a parallel CUDA based multi-colony ACO for solving the symmetric traveling salesman problem (TSP). Your code should output the path as a sequence of node numbers and the minimum path length found. You should do NONE of the compution on the CPU. The only CPU functionality you should need is maybe print line. All the other functions called in CPU code should be GPU related (ie. functions beginning with cuda*).

A simplified description of the algorithm for a single colony is below.

1. Initialize all edges to a small initial pheremone level (τ_0)
2. Initialize each ant to a randomly chosen city
3. For some number of time steps / iterations:

(a) For each ant, build a tour T by applying the probabilistic transition rule

(b) For each edge E, apply the pheremone update step

Where the probabilistic transition rule determines how an ant chooses its next city. Each ant, k maintains a tabu list of cities it has already visited an will never visit again. The probability probability of an ant traveling from city $i \to j$ at time t is

$$p_{i,j}^{k}(t) = \frac{[\tau_{i,j}(t)]^{\alpha}\,[\eta_{i,j}]^{\beta}\,\left[\mathbf{1}_{j \notin T^k}\right]}{\sum_{l \in J_i^k}\left([\tau_{i,l}(t)]^{\alpha}\,[\eta_{i,l}]^{\beta}\right)}$$

Where α and β are tunable parameters controlling how likely an ant is to explore or follow a path. $\tau_{i,j}$ is the amount of pheremone on the path from node i to j. $\eta_{i,j}$ is the visibility heuristic between cities i and j (you can just use the inverse distance between the two cities). J_i^k is the set of cities that ant k still needs to visit (not in the tabu list). And $\mathbf{1}_{j \in T^k}$ is the indicator function, which has a value of one if city j is not in the tabu list and zero if the city is in the tabu list.

The pheremone update step simulates pheremone evaporation and is given by

$$\begin{aligned}\tau_{i,j}(t+1) &= (1-\rho)\tau_{i,j}\sum_{k=1}^{m}\Delta\tau_{i,j}^{k}(t)\\ \Delta\tau_{i,j}^{k}(t) &= \begin{cases} Q/L^k(t) & \text{for each edge } (i,j) \text{visited by ant } k \text{in iteration } t\\ 0 & \text{else}\end{cases}\end{aligned}$$

Where $\rho \in (0,1)$ controls the speed of evaporation and Q controls how much pheremone is put down. $L^k(t)$ is the length of the tour T of ant k at time t. m is the number of ants in the colony.

For your multi colony configuration, you only need return the minimum path found. There are other (better ways) of utilizing multiple colonies (ie. sharing pheremone after some number of iterations), but these are not required.

You must use muli-thread reduction when performing operations across threads, ie. the following is NOT ok:

```
// bad code example
__shared__ int cache[];
cache[threadIdx.x] = threadIdx.x;
__syncthreads();
if(threadIdx.x == 0)
{
        for(int i=1; i<blockDim.x; i++)
        {
                cache[0] += cache[i];
        }
        printf("%d\n", cache[0]);
}
```

3 Useful Information

We have download TSPLIB, a library for TSP problems containing several TSP problems and their solutions. These can be found in `/course/cs195v/tsp/tsp/`. You can easily view the best solutions for all the problems at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html. A good sanity check is to make sure your implementation never performs better than the optimal solution[1].

In the support code, we have provided a barebones parser for reading in the .tsp files. It only supports TSP problems which have edge lengths defined as EUC_2D (Euclidean 2D distance) or GEO (geo distance) keep in mind that the formula for calculating distance will depend on which type of graph it is!

For computing geographical distance, you should use

$$\begin{aligned}
RRR &= 6378.388 \\
q1 &= \cos(longitude[i] - longitude[j]) \\
q2 &= \cos(latitude[i] - latitude[j]) \\
q3 &= \cos(latitude[i] + latitude[j]) \\
d_{ij} &= RRR(\arccos(0.5((1.0 + q1)q2 - (1.0 - q1)q3))) + 1.0
\end{aligned}$$

Don't forget to fill out TspEdge.dist which is unitialized (you probably need to write some sort of init kernel).

Support code can be found in `/course/cs195v/aco/`.

4 Notes

We suggest that you split your colonies into blocks (one thread block per colony), with each thread corresponding to a single ant.

Be careful when using shared memory, there is only about 48kb of shared memory available to each SP it's very easy to overrun this limit if you try to store pheremone data in each SP as shared memory, in which case the GPU will fail.

It might make it easier if the number of ants / threads per colony / block is power of two (especially for reductions).

Because the GPU is driving both X and CUDA, a kernel which takes too long to complete will lock up your display, and may be killed automatically by the X server.

Be careful with floating point precision. η, the city visibility can be very small if cities are relatively far aprart. When raising this to a power greater than one, you may have precision errors if you're not careful.

[1]If it does, something is wrong.