Parallel Ant Colony Optimization Using CUDA

Octavian Nitica
Research Advisor: John Cavazos
Department of Computer and Information Sciences
University of Delaware High Performance Computing Fellow

Abstract

The Ant Colony Optimization (ACO) Algorithm is a metaheuristic that is used to find shortest paths in graphs. Because of the structure of the ACO algorithm it is amenable to being parallelized. By using CUDA to implement a ACO algorithm, we achieved significant improvement in performance over a highly-tuned sequential CPU implementation. We also found that we could run multiple ant colonies in parallel. Running more ant colony simulations allowed obtain a better distribution of results, which resulted in a better overall solution. The construction step of the ACO algorithm consists of each ant creating an independent solution, and this step is where most of the computation is spent. Since the construction step is the same for most ACO variations, parallelizing this step will also allow for easy adaptation to different pheromone updating functions. Currently, our research tests this hypothesis on the travelling salesmen problem (TSP).

Introduction to ACO Algorithm

The ACO algorithm gets its name from real ants, because it models the way ants search for food. Ants in the real world begin by randomly searching for food. As ants find sources of food, they leave pheromone trails that allow other ants to find the food. Over time, ants converge to the nearest food source, because it has the strongest pheromone trail. Pheromones also evaporate over time, which is important so ants do not continue to go to the same food source after it has disappeared. The ACO algorithm works in a similar way. It has two main stages: the construction phase and the pheromone update phase. In the construction phase, individual ants construct a sample solution to the problem using a probabilistic function. The function uses a heuristic function and the amount of pheromone on the edges to decide which city to choose next. Once all ants have constructed their respective solution, the ants enter the next phase. In the pheromone update phase, certain solutions (generally the best ones), deposit pheromones on the edges of their solutions. Also, old pheromone trails have their potency decreased during this stage to prevent early convergence to a suboptimal solution. After a certain time period, ants will converge to a near-optimal path through the graph.

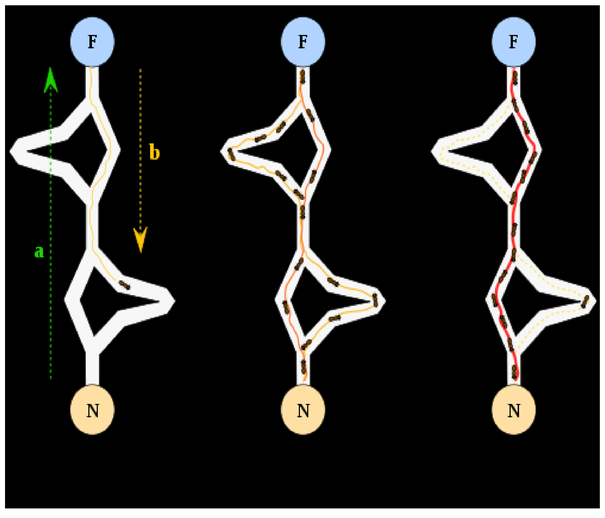
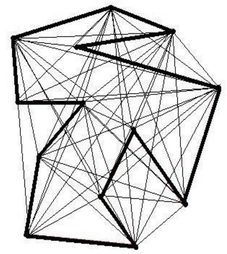


Figure 1. Example of ants searching for food.

Photo used under Creative Commons from http://en.wikipedia.org/wiki/File:Aco_branches.svg

Travelling Salesmen Problem

The travelling salesmen problem is a classic computer science problem. It is often used as a sample problem for graph search algorithms, due to the fact that it is easy to explain and understand. The problem consists of a graph where each node is a city. Each city has weighted edges connected to every other city. The problem is to find the shortest path through all the cities. This problem was chosen for several reasons. For one, there is lots of research done in this area, and their are problem sets of available that include sample city sets and solutions. Also, since the nodes are strongly connected and symmetrical, it requires less checking when choosing the next city. Finally, there already existed highly-tuned CPU ACO sample code that could be modified and compared against.



Platform and Result Graphs

Server Specifications:
- CPU: Intel Xeon E5335 @ 2.00Ghz
- GPU: Nvidia 8800GTX 1GB (128 stream processors)
- System Memory: 2GB

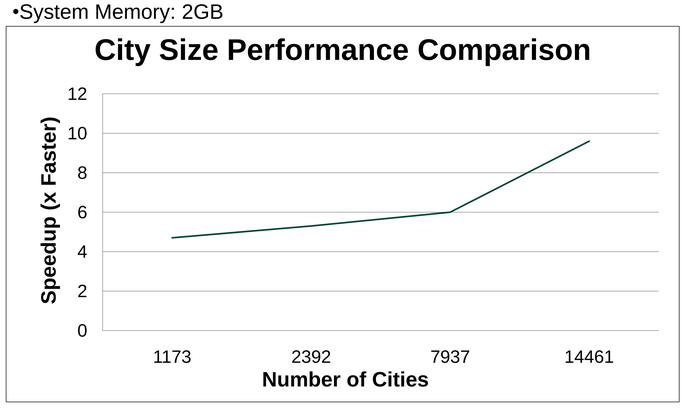


Figure 2. Comparison of CUDA code speedup versus the sequential version as city size is increased.

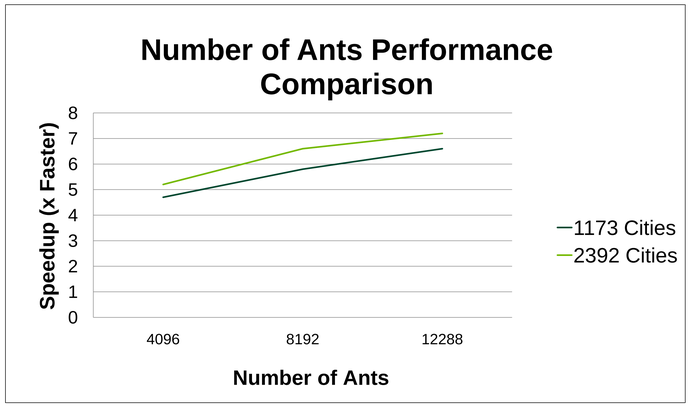


Figure 3. Comparison of CUDA code speedup versus the sequential version as the number of ants is increased.

Observations on Results

- Problem specific values:
- Alpha – 1 (influence of heuristic information)
- Beta – 5 (influence of pheromone trails)
- Rho – 0.5 (pheromone evaporation rate)
- N – Number of cities (varies)
- N_Ants – Number of ants (varies)

The results show a definite improvement in execution time when using our CUDA implementation over the CPU implementation on large sized graphs and a high number of ants. As we increase the problem size (i.e., number of cities) or number of ants, the performance gap between the CPU implementation and the CUDA implementation also increases. Since the construction step in the ACO algorithm is general and applicable to many pheromone update functions, these speedups are achievable for several variations of the ACO algorithm. Some of the variations include Max/Min, Elitist, Ant System, and Rank Based System. It is important to note that the solution qualities for the CUDA implementation were equal to the CPU implementation for small problems and better for larger problems.

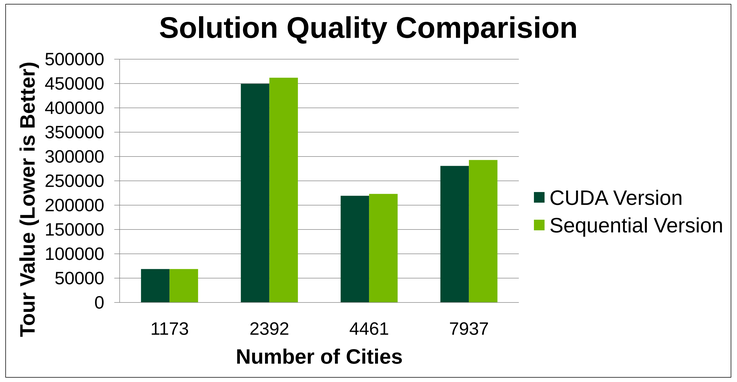


Figure 4. Comparison of solution quality between the two versions of code. The 7937 city example tour value has been divided by 100 to allow for better scaling.

Conclusions

Our initial research results are promising. There is significant speedup when running larger problem sizes on the GPU versus the CPU, and we believe a much greater speedup can be achieved once the entire algorithm is ported to the GPU. The speedup should be enough to allow several colonies to run at once, allowing for much more potential improvements to the algorithm. The speedup already gained is significant for larger simulations, taking a normal number of iterations down to under five minutes from thirty minutes in some cases.

Acknowledgements

- Thomas Stuetzle. ACOTSP, Version 1.0. Available from http://www.aco-metaheuristic.org/aco-code, 2004.
- University of Delaware PetaApps Cloud Physics Group for funding my research