Rob Fitzgerald
CSCI 7551 Parallel
Final Project Report

Traffic Assignment

This project was an attempt to build a vehicle traffic simulation application built in C++ using OpenMP. The technique described in [1] describes a gradient descent algorithm which iteratively applies flows to the traffic network. For each flow of traffic from an origin to a destination, a shortest path search is calculated based on the current (partial) costs, and then the flows are re-assigned. The shortest path calculation was implemented using a bidirectional a* approach [2], with a bidirectional heuristic function which attempts to find the best new node for both search directions simultaneously, by computing a cross product of all node distances on the frontier of both searches.

I was unable to complete a fully functional version of the application by this deadline. That said, I have written all code for all parts of the search and simulation, and have solved many problems that have emerged in the process. I will share a few notable examples.

A major challenge was the discovery of the behavior of scope in OpenMP. Many functions were working fine, but one in particular was causing segmentation faults. I looked for many hours for ideas on the web that might clarify. Eventually, I realized that this particular function was a global, not a member of one of my shared classes. Even though it was not a data structure, it was still living out-of-scope of any child process, and therefore it would fail.

Another issue which has been a greater issue than expected is the synchronization of my process pairs. In my implementation, two processes are spawned for each search on an origin/destination pair. Both processes share the same code, but one (the one with the even job ID) is treated as the master. Both will advance their own search, but then only the master will evaluate the state of the search and find the best-fit frontier nodes for continuing the search. I thought critical sections would be enough to synchronize these pairs, but I found I needed to manage synchronization at a much finer level for things to “work”, so I implemented a spin-wait which waits for a flag from the conjoined process to be flipped.

I felt this project was great practice with a problem I always want to improve on, which is working with graph algorithms. This implementation uses pointers to build a graph of memory associations, instead of using an adjacency matrix or other graph representations. I enjoy working in these structures where the form of the implementation is consistent with the real-world problem it emulates. I chose OpenMP because I found it’s API to be fairly simple, friendly and logical. I did enjoy working with it.

My hope is to continue this project through the winter. When I get the searches tied down and run successful simulations, I will add the code to read in origin-destination pairs from the input data files. I will then implement the Frank-Wolfe algorithm which is an optimized version of the algorithm I am using. I will also find ways to reduce the complexity of the implementation, by removing unnecessary iterations, copies, and trim the fat in general.

References

[1] Juan de Dios Ortuzar, Luis G. Willumsen, "Assignment" in *Modelling Transport*, 4th Ed. New York: Wiley, 2011, pp. 349-390.

[2] http://theory.stanford.edu/~amitp/GameProgramming/Variations.html