The Search for an Efficient Parallel Implementation for Calculating Total Viewsheds
Tucker Gordon, Class of 2017

My summer research involved a more and more frequently occurring problem in computing: what do you do when your problem is too big for one computer? In my case, the problem was computing a total viewshed. Given terrain data, and a point on that terrain, a viewshed is every point visible from that point. A total viewshed means doing this calculation for every point. When the terrain consists of millions, or even billions of points, this calculation can take one computer almost a week to do. In order to solve this problem, one must turn to parallel computing: distributing parts of the overall job equally to many different processors.

After learning Message Passing Interface (MPI), a language used for parallel computing, I began testing a parallel implementation (using Bowdoin’s High Performance Computing Grid) for this problem that Daniel Cohen ’15 created the previous year. His first idea was to divide the terrain into equal size chunks, and give each processor one chunk. Once they had done the viewshed computations, all chunks would be reconciled into the output. While this worked in speeding up the computation, it was clear that it could be more efficient. Terrains often contain some points without elevation data (i.e. water), and when a computer finds one of these points, it can immediately move on to the next one. Therefore, if a processor’s chunk has lots of these points (which happens with large bodies of water), it will finish well before the other processors. This is inefficient, as it means some processors wait around idly while others continue working.

Daniel came up with a few solutions to this problem in his time, that I tried to implement, but all fell short one way or another. One idea was to give a few points to all but one processor to start, and then when they finished, they would signal the “master” processor, who would then give them more work. The problem however, was that the master processor spent most of its time not doing anything, which is a waste of a lot of computing power.

A second idea was to give each of the P processors every Pth point. In theory, this would make it so that each processor sees almost an identical array of points, and thus finish around the same time. However, the MPI library did not have a way of implementing this, though it could be looked at more in the future.

Professor Toma and I came up with a third method though that would accomplish the goal of identical chunks of points, in a way that could be implemented with MPI functionality. Each processor takes K elements to compute viewsheds for. Once all have finished, their results are sent to master array, and then they each take K more elements. This process continues until all points have been calculated. The result is that all processors see a very similar assortment of points, and therefore finish around the same time.

After implementing this algorithm, I began testing it to determine how efficient it was. One measurement of efficiency is to look at the standard deviation for the visibility calculation times of individual processors. A small number means they all finished closely together (efficient), and a large number means they finished at very different times (not efficient). The times of the original algorithm, in which all processors worked with a single equal-sized chunk, had a standard deviation of 10.43. The algorithm that we created had a standard deviation of only 2.37. This was a very encouraging result, and it allowed us to calculate total viewsheds much more efficiently. While there are still areas that could be improved, the result of my research this summer was an algorithm that was over 4x more efficient than any we had before.

Faculty Mentor: Professor Laura Toma
Funded by the Maine Space Grant Consortium
Bowdoin College is an affiliate of the Maine Space Consortium