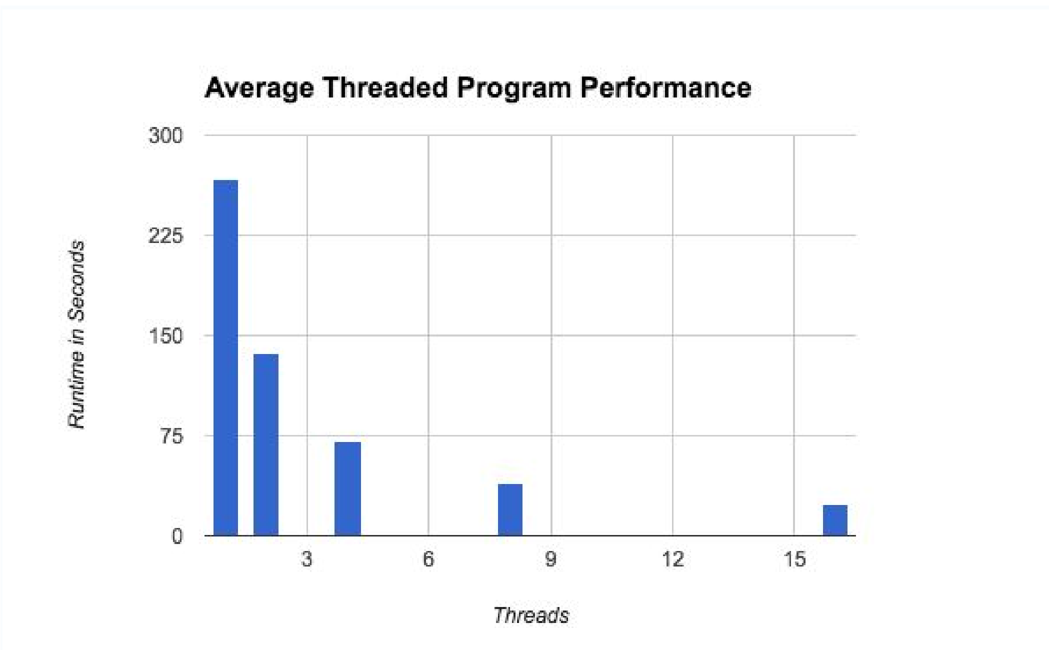
Phase 7 - OpenMP

IT 515R - Scientific Computing
Tanner Satchwell

My approach in using OpenMP in my solver program was to implement it with as little change to my code as I could. This was a little difficult to do because, as we discussed in class, you don't spread out #pragma omp commands through functions. Generally best practice is to combine those functions so all of your #pragma omp code is in close proximity. Because I didn't want to change my code very much, I didn't follow best practice in this regard. This made it somewhat difficult in some instances to decide where I should put barriers to avoid deadlock and race conditions.

One change that I did end up making to my code was the way I checked to see if the grid was stable. Prior to implementing OpenMP, the isStable function returned as soon as the error was greater than epsilon. Because you can't break out of a for loop like that when you are using multiple threads, I had to move the error check outside of the for loop. This way, I was able to do a max reduction on the for loop, and then check if it was greater than epsilon afterward.

I was pleasantly surprised at how much of a performance increase using more threads added. Running my openmp program with a single thread took nearly five minutes, but adding more threads reduced that time by about half every time I doubled the thread count. See the following graph for a more detailed comparison.



Overall, I was pleased with the increased speed the program ran with more threads. If I were to do this project again however, I would probably do some major restructuring of my code to make threads work more intuitively. As it is, I'm still not 100% sure there are no race conditions. In Phase 8, I will most definitely restructure the code to make implementation of c++ threads easier.