I am Bruce Darcy, an undergraduate senior in Comp Sci, and I will be working with the back propagation algorithm.

I am choosing back propagation and neural networks, because they are becoming more and more widely used for various applications in modern day processes, ranging from oil rig sensors with IOT monitor data processing at Shell, to recommending netflix videos to users.

For my project, I will be looking at parallelizing back-propagation and Neural Networks code. My code comes from an open source C repository on GitHub that allows building of classification networks, and as of now all the code is sequential only. I hope to contribute my work with parallelizing this code to the open source project when I'm done.

Link to the repo: https://github.com/codeplea/genann

I will be using an IRS data set to train the model and evaluate speedup. Here are some baseline timing for various amounts of training iterations and network hidden layer sizes. All networks contain 4 inputs, 4 hidden layers with N nodes in them, and 3 outputs.

250 nodes 100 iterations:	9.436s
2500 nodes 10 iterations:	2m38.114s
25 nodes 1000 iterations:	0.693s
25 nodes 10000 iterations:	6.610s
5 nodes 1000000 iterations:	46.687s

I've profiled the code will callgrind and cachegrind. There are very few cache misses, less than .1% on the smallest cache, so no blocking or other optimization is going needed there.

From callgrind, I can see that about 4 million out of 11 million of the cycles are spend on the forward pass through the network, and the remaining 7 million are mostly spent on the backward pass through the network.

Back propagation pushes the values of the edges in the neural network towards a local minimum through gradient descent. First a forward pass is one through the network to see what the current output is, and the amount of error that is in that output compared to what it should be. Back propagation then calculates the error in the output, then assigns each neuron a "contribution" error value corresponding to how much it contributed to the final error. Then each neurons output is adjusted a bit towards producing less error, a process known as gradient decent. This process is then repeated a many times until the neural network is working well.

With this particular implementation, I would like to shoot for a decent 2x-6x speedup using OMP, then move the problem to Cuda on a graphics card, or do the problem with MPI. Back propagation should ideally be run on a graphics card, but I may choose MPI instead if it seems to be an interesting and feasible experiment. Also maybe some combination of both, using a distributed network of machines with graphics cards to do back propagation. MPI is probably going to be most feasible for very large network sizes, where one pass through back propagation takes a long amount of time.