Brian Crafton
High Performance Computing
February 8, 2016

Project Proposal: Parallelizing A Neural Network

Neural networks are a popular form on machine learning used for classifying data. They use a forward and back propagation technique to compute a gradient and a cost. The gradient is a computed value of the direction of the solution and the cost is the scaled "failure" of the current solution. Using these two values computed using forward and back propagation an optimization method can find a local minimum solution. By fine tuning the value for the learning parameter the programmer can hit a good local minimum rather than one that is not a very good solution. Machine learning uses statistics to give accurate predictions based on training data. A neural network is a supervised, discrete implementation of machine learning. Supervised machine learning uses training data that has already been classified. Discrete machine learning classifies the input data as a class or category as opposed to giving a continuous prediction.

Neural networks require lots of matrix arithmetic and lots of repetitive steps. Matrix arithmetic requires lots of independent instructions, making it ideal for parallel computing. Operations like matrix addition, matrix transpose, matrix multiplication, matrix splitting, ect are all computationally intensive operations that have lots of (embarrassingly) parallel operations. This type of parallelism is data parallelism because the same computation is being done just on different data. To exploit data parallelism I plan to use SIMD/SIMT and multi-threading.

The application for this project is classifying hand written digits, so taking 20x20 pixel (greyscale, uncompressed) images and classifying each one as a digit (0-9). Luckily this is a common problem and there is tons of training data available online. The application for this project is based on an assignment from the machine learning course available at *Coursera.com*. In the online assignment it was written in Matlab, but the implementation for this project will be in C, specifically written for the gnu gcc c compiler. There will be two intentional design flaws from the start.

1. I will be writing basic implementations for the matrix arithmetic from scratch
2. I will be writing and using gradient descent rather than using a popular optimization function

The reason for these design choices is because the assignment is about learning rather than building an effective neural network. Success will be determined by the parallel speed up achieved rather than the actual accuracy of the neural network. There are two high level techniques that will be investigated.

1. Paralleling the matrix arithmetic
2. Giving each thread its own neural network iteration in the neural network and have each thread perform sequentially and come back together to compute the new classifier.

There will be tradeoffs between both, but after doing some trial and error I am hoping that the better technique will be clear.