Parallelizing Neural Network

CP341 - Parallel Programming

Mishig Davaadorj
mishig.davaadorj@coloradocollege.edu

I. Abstract

Theoretically, parallelizing neural networks can give us a speedup of multiple magnitudes as neural networks extensively use matrices and matrix operations. Due to their nature on how they are stored on processors, matrices offer increased data locality and exciting opportunity for SIMD (single instruction multiple data) parallelism. However, practice of parallelizing neural networks outputs mixed results. Parallelized version of the neural networks trained slower due to potential issues like overhead required for parallelization consuming more time than the actual parallel processes. Nonetheless, parallelized version of the neural networks demonstrates speedup in testing/prediction and individual matrix operations.

II. Introduction

Recently, neural networks have become a hot research area for its ability to construct complex models that are capable of producing great results. For instance, artificial intelligence research areas such as natural language processing and computer vision were able to achieve human-level abilities through neural networks (LeCun 436). Moreover, Cybenko theoretically proved that neural networks with one hidden layer is a universal approximator (303). From a mathematically perspective, neural networks is a series of linear transformations with a non-linear function (sigmoid, tanh, etc.) being applied in between the layers. Matrices are used as building blocks for neural networks. Therefore, most matrix operations (matrix algebra) can benefit greatly from parallelism as matrix operations are usually applied element-wise, which means those individual operations are independent from one another. Specifically, matrix addition, subtraction, element-wise multiplication, and any function that acts on matrix element-wise can be parallelized with Map pattern as those operations can be classified as embarrassingly parallel problems. Another big matrix operations that is crucial in neural networks is matrix multiplication, which transforms values from one layer into another, and matrix multiplication can be parallelized with Reduce pattern as some kind of summing process is involved. This paper presents a neural networks with one hidden layer that is trained to predict whether a voice is of a male or female person. Both serial and parallel versions of the program are implemented.

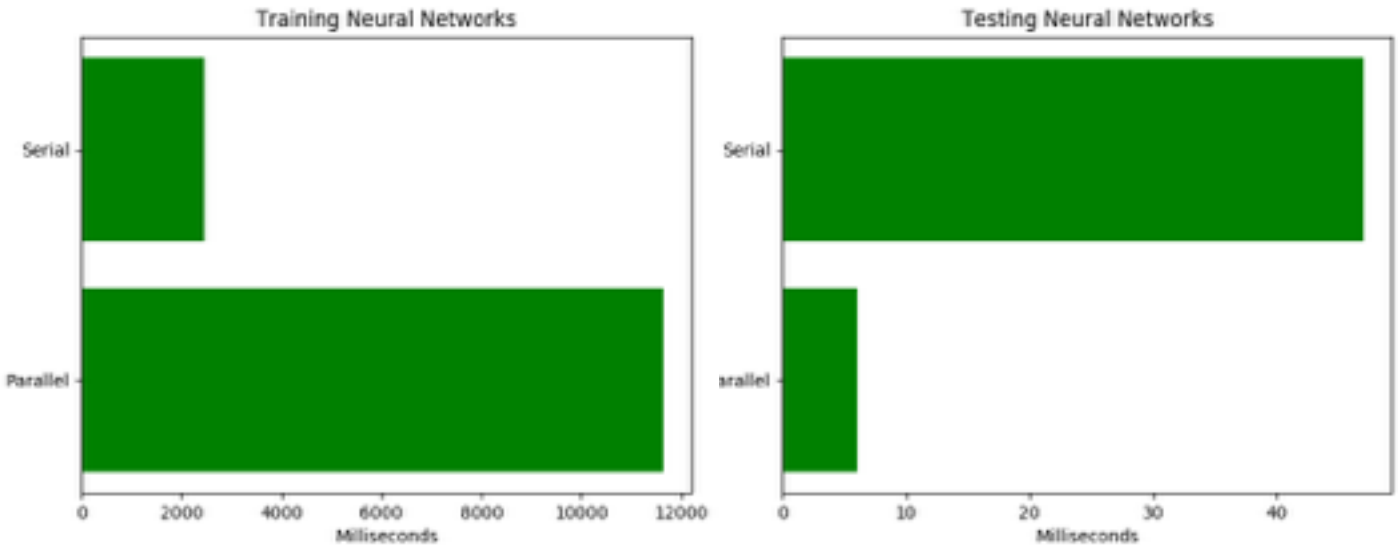
III. Design

The program consists of three classes: Mat, Dataset, and NeuralNetwork. Dataset class offers a convenient way to load CSV files into C++ arrays, and Mat class implements matrix and matrix operations. Mat class uses one-dimensional arrays that are created on the heap to abstract the logic of matrices since big matrices and arrays can not fit on the stack. NeuralNetwork class heavily uses Mat class to train and predict.

Parallelizations are done in Mat and NeuralNetwork classes, and OpenMP was my choice of technology for parallelization. Six operations in Mat class were parallelized. Four of them (addition, subtraction, element-wise multiplication, generation of random matrices) use `#pragma omp parallel for` as they operate on one-dimensional arrays. Transpose operation consists of two nested for loops, therefore uses special OpenMP notation `#pragma omp parallel for collapse(2)`, telling the compiler to parallelize both outer and inner for loops. All those five operations implement Map pattern. Unlike those operations, matrix multiplication implements Reduce pattern as entries of different columns and rows of two matrices must be summed (dot product). OpenMP provides an easy way to implement Reduce pattern with: `#pragma omp parallel for reduction (+:your_variable)`.
There is no more operations to parallelize during the training phase of the neural network as some parts like weight updating is supposed to be serial. However, the entire testing phase can be parallelized. Map pattern made it possible to parallelize testing multiple inputs to the model at the same time, and Reduce pattern was implemented to count how many of the predictions were correct and incorrect.

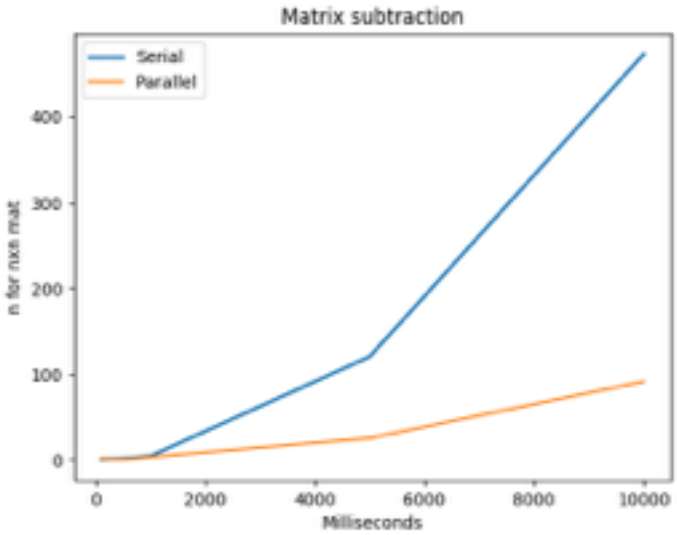
IV. Performance Analysis

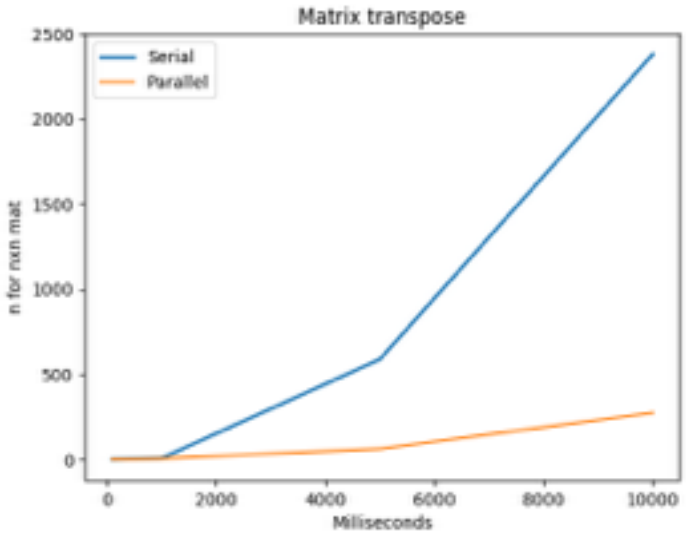
The neural network was trained to recognize gender from voice data, and the best accuracy the neural network achieved was 98%. The network consist of three layers: input (Mat 1x100), hidden (Mat 100x100), and output (Mat 1x1).
In terms of parallelism, the program gives mixed results. Training process of the network on the parallel version is 4X slower than that of on the serial one. However, testing process of the network on the parallel version is 7X faster than that of on the serial version. Also, individual parallelized matrix operations demonstrate much superior performance, compared to their non-parallel counterparts.



One conjecture why parallel neural networks example was much slower than its serial version is the fact that the network consisted of relatively small matrices of sizes 1x100, 100x100, and 1x1 and the overhead of starting parallel processes on these matrices

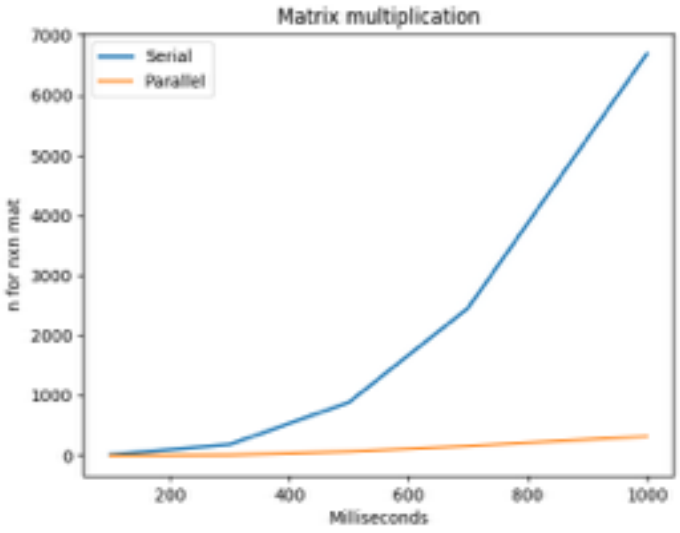
took more time than the actual parallelized processes. 7X speedup shown in the testing phase backs up the conjecture as it parallelized a much bigger array of 634 elements.





To gain a deeper understanding on the performance of my Mat class, I timed specific operations. I divide the operations into three types:

I. with one for loop (subtract, add, element-wise multiplication operations)
II. with two for loops (transpose operation)
III. with three for loops (matrix multiplication operation)



The results demonstrate that there is no speedup when matrix size is small (1000x1000 for type I and II operations and 100x100 for type III operation). However, as the size of the matrix increases, speedup increase exponentially. This exponential speedup is expected as the parallel processes are operating on arrays of many elements, which provide a good data locality.

V. Conclusion

Despite individual matrix operations and testing phase of the neural network achieved speedups through parallelism, the training part did not. The following suggestions might improve the speedup on the training phase:

1. train a much complex model like object detection (images will create much bigger matrices than CSV file of voice attributes data the network used for this paper).
2. implement parallelism in CUDA or OpenCL as GPU are designed to parallelize linear algebra operations.
3. parallelize through different more complex patterns rather than using simple Map and Reduce patters as done in this paper.
4. implement a training process that uses mini-batches to parallelize more processes.

References

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015)
2. Cybenko, G. Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems, 2 (4), 303-314 (1989)