Parallelization of Convolutional Layers

Zach Josten

1 Abstract

Kernel matrices are an important part of filtering processes, allowing application of different "functions" onto each element present in whatever it is that one may be applying it onto. It is no different for neural networks and their convolutional layers, where all of the features of a previous layer are gathered and aggregated into each feature of the new layer. This allows neural networks to learn quickly and deeply. However, this requires an incredible amount of operations to be made upon the data. But as will be seen, the speed of these can improved by splitting up the workload.

2 Problem

Convolution layers make use of kernel matrices and matrices of features in order to train, apply weightings and transform the feature maps in order to work best for the next layer and this allows great training of neural network models. The problem as mentioned before lies in the many different operations and how they overlay one another, meaning that an operation is done many times over due to it having to be done on all things for numerous amounts of times. It can be better seen in the equation for computing the required number of add operations, we use M to represent number of features in a layer, W and D for the dimensions of each feature, K for the dimensions of the kernel matrices used, W' and D' for the new output feature dimensions and N for the number of output features.
$(K * K * M * N) * W' * D'$
(3 * 3 * 128 * 64) * 32 * 32
= 75, 497, 472
This is a relatively small example and we still get a very large amount of operations. Later on we will use a bigger example with the total number of operations coming out to around 4.8 billion.
The reason that this occurs is because for each kernel matrix we need, we have to apply the kernel matrix onto every element in each input feature, for every input feature and then aggregate the results.

3 Methods

The methods here are nothing too crazy. Basically the idea is to parallelize across kernel matrices. We do this by splitting up the kernel matrices among the processors we have(therefore requiring the kernel matrices to both be more than and divisible by the number of processors), then doing their own respective computations in parallel. The one small thing is in order to make the matrix struct that has been used throughout the semester play nice with MPI(sending it over MPI recv and send), we must first construct a new MPI datatype, this can be seen at the top of the code(this was done simply by looking through MPI documentation). Other than that I have recycled some of the code used in class for a basic structuring of MPI sending and receiving. Each processor then has the function aggregate and convolute. Convolute performs the iteration of elements per input feature per kernel matrix it is in charge of and does this by calling aggregate on each element, which simply does an application of a kernel matrix onto the current element. Overall simple in idea, but was a bit of work to get implemented and bug free.

4 Results and Conclusion

This leads us to the results, which to put it simply, were very good. In the presentation attached, we can see that there was only improvement in both cases. Starting with the small case, we can see that from the 1 processor case and 32 processor case there was a 30.41 times improvement. With the bigger case we can see a 30.89 times improvement, which is slightly better than the smaller case. This is incredible, because it comes out to a nearly linear improvement with respect to processors. In the bigger example, the 1 processor case of 13.837 seconds to the 32 processor case of 0.448 seconds overall improvement is very nice to both see and look at. The only other thing I would interested in is seeing if we could use the threads of each processor to go over specific input features and have those done in parallel as well. Also to see it go through multiple layers, in order to pseudo-simulate a neural network would be cool, but most likely would see similar results to the current results as it would just be doing the same thing over a few times more.
In conclusion, the project is a success, improvements were made and over some very large amounts of operations and I am quite satisfied with it.