ImageNet Classification with Deep Convolutional Neural Networks

Problem analysis

October 6, 2016

Introduction

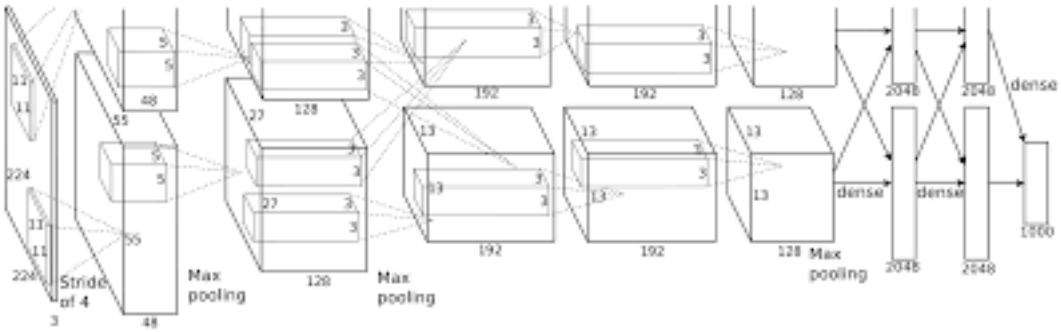
The goal of our project is to develop a parallel CNN using OpenMP starting from the neural network developped by three students of the University of Toronto. They have created a big CNN exploiting CUDA architecture with the power of two parallel NVIDIA GPUs, so the time for training the net on a large amount of images taken from ImageNet is decreased and the error rate is improved. In the next paragraphs we will describe the architecture of the CNN with all the main characteristics and the requirements, the specifications and the implementation methods to trasform it using OpenMP, explaining all the possible issues and the differences between the two techniques.

Architecture of the network

The CNN developed for this project is a large network having 8 layers: the first 5 are convolutional and the last 3 are fully-connected. The convolutional layer extracts features using a set of kernels which, convolving on the previous layer, computes the dot product between the entries of the kernel and the input. This layer is often followed by the max-pooling layer. It partitions the image into a set of non-overlapping rectangles and outputs the maximum value from each sub-region. In this CNN, instead of using non-overlapping, they change it with overlapping pooling, because it reduces the error rate of 0.4% and 0.3%. The fully-connected layers are basically the same layers used in the normal neural networks. The CNN receives in input an image of dimension 224x224 (RGB values for each pixel)extracted from the original image of size 256x256. The images used to train the network come from ImageNet, which is a database of over 15 million labeled high-resolution images of almost 22000 categories. All of these images have several different resolutions, but they have been down-sampled to a fixed resolution of 256x256. The principal problem in NN is overfitting: when the model fits the noise in the data. To solve this problem, they used a unique technique called "dropout". It sets to zero the output of each hidden neuron with probability 0.5, so that the complex co-adaptations of neurons is reduced. Also a local response normalization scheme is used to improve generalization and precision of the prediction (by around 1.5%): Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression:

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-\frac{n}{2})}^{min(N-1,i+\frac{n}{2})} \left(a^j_{x,y}\right)^2\right)^\beta}$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer.



Multiple GPUs

The principal innovation of the CNN provided by these guys from the University of Toronto is the parallelization of the network using CUDA with two NVIDIA GPUs. They wrote a highly-optimized GPU implementation of all the operations required for the training phase. They run the neural network on two GTX 580 3GB GPUs by putting half of the neurons on each one. As we can see in the architecture of the CNN in the picture above, the kernels of layer 3 take input from all the kernels of layer 2, so the GPUs communicate only between these two layers. In all the other layers, the kernels take input only from the previous kernels that reside on the same GPU. Thanks to this parallelization scheme, the error rates are reduced by 1.7% and 1.2% as compared with a neural network having half of the kernels trained on one GPU. The training time is between five and six days, which is lesser than training data on one-GPU. Another important aspect is that in every run the two GPUs had different specialization. The kernels of the first GPU were color-agnostic, while the kernels of the second GPU were color-specific. This happened independently by any weight initialization. This CNN with all the characteristics describes so far, has been trained on the subset of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions and achieves top-1 and top-5 test set error rates of 37.5% and 17.0%, while the best performance achieved during the ILSVRC-2010 competition was 47.1% and 28.2%, which means that this scheme has an error rate of almost 10% better.

Development using OpenMP

OpenMP offers an easy alternative to CUDA to implement this parallel application. Since the parallelization of the training of this type of neural network achieved not only a reduction of computation time, but also a different specialization distributed amongst different parts of the network.This different specialization of features recognition was achieved thanks to the architecture of the layers and the different policies of communications between the two GPUs on which the network was trained. With OpenMP and a machine with at least 2 available cores, the final capability of the network to recognize images should

be exactly the same. However the training time should be much worse given the fact that GPUs are specialized in SIMD (and have more cores to handle the data) while CPUs are not. We also believe that by changing the machine on which the network is trained with one with more cores, is possible, with OpenMP, to specialize more areas of the network to recognize specifically more different features and thus increasing the accuracy of the prediction at the cost of an increased training time. The increased cost for a network as large as the one described in the paper could be not trivial. Implementation with OpenMP of the network and the algorithm should be pretty straightforward by assigning half of the network training to a thread and the rest on the other. Communication between layers on different thread could be done with barriers and shared attributes. Given the languages supported by OpenMP we believe that C is the best option to implement neurons and layers functions since we only need mathematical functions. The size of the network and the dataset are huge so we will work with a reduced version of the CNN and of the input ImageNet set.

Advantages and disadvantages of an OpenMP implementation

OpenMP offers somegreat advantages with respect tothe application we have to deploy:

- Easy to implement;
- Memory sharing (easy communication between layers);
- Directives can be added incrementally;
- Code can still be executed sequentially;
- Easy to understand the code.

Some disadvantages with OpenMP are:

- The application could scale poorly on different machines;
- Focus on loop parallelization;
- Training time increased.

Conclusions

The implementation of the CNN presented is feasible with C and OpenMP. For the sake of the project we will use a reduced set of training images (taken by ImageNet) and also a reduced number of neurons for each layer. We will split the training on two cores and use the same policies of communication between different layers of the original network. The input images will be simply scaled down to a fixed resolution and then each pixel data (RGB) will be fed to the network.