CS486 Project: Sum Product Networks on GPU using OpenCL

Ruiming Zhu, 20413389

Introduction

A sum product network (SPN) is a deep neural network composed of sum and product nodes. The main advantage of a sum product network is that it allows for tractable inference. OpenCL is a framework that allows someone to write programs which run on highly parallel hardware, such as GPUs, multicore CPUs, and even FPGAs. In this project, I propose a expectation-maximization based learning algorithm for training weights of sum product networks on the GPU using OpenCL.

Motivation for OpenCL implementation

While SPNs allows for tractable inference, it can still take a long time to learn weights on a SPN on larger models. GPUs have been used to great effect in machine learning to parallelize and speed up the learning process. SPNs have been trained on large distributed system of computers using the message passing framework MPJ Express. Subsequently, work was put into learning SPNs on powerful NVidia graphics cards using NVidia's CUDA framework, which is another framework for writing programs for the GPU. CUDA and OpenCL are similar. One distinction between the two is that CUDA is designed to only run on NVidia graphics cards, while OpenCL is designed to work with any GPU. This opens the gateway to many applications, including potentially using this algorithm on average consumer computers or mobile devices, which can use their own built in graphics hardware to drastically increase performance in learning SPNs.

OpenCL architecture: OpenCL tasks are split into several threads

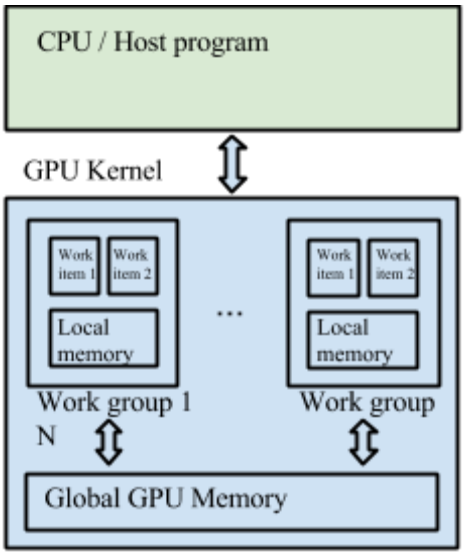

Figure 1: Architecture of OpenCL programming

Due to the highly parallel nature of the GPU, the method of structuring programs will be slightly different than a regular program. An OpenCL program is comprised of two main parts: the host program, which runs on the CPU, and the kernel program, which runs on the GPU. The host program will be primarily used for configuring the GPU and sending work to the GPU. The kernel program will utilize the parallelism of the GPU to do the bulk of the computation.

In the OpenCL architecture, there are several work groups, each comprised of a number of work items. Work items are the basic compute units and will each execute the kernel program. Work items within a single work group share local memory and can be synchronized with each other. Work items in different work groups can not be synchronized with each other. As a result, work done in each work group needs to be independent of each other.

Sum Product Network representation

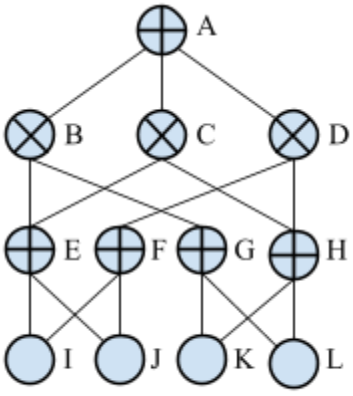

Figure 2: Example of sum-product network.

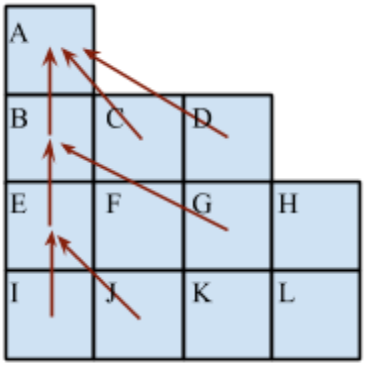

Figure 3: Computational matrix representation of *Figure 1*

A SPN is comprised of sum nodes and product nodes. To perform inference, we take the variable nodes which are in evidence and replace their probabilities with 1 or 0, depending on their observed value. Then we go through an upwards pass, either summing or multiplying the weighted probabilities at each sum or product node. We can represent this as a computation matrix, where each row in the matrix represents a row in the sum product network. To compute inference in this matrix, we set the observed variables as usual. Then starting from the last row, we will compute all the values of the row using values of the rows below it. As long as the computations are done within one work group, the work items can synchronize with each other and ensure that all calculations in a row are completed before going on to the next row. The process for calculating the MAP or performing EM is similar. These algorithm either starts from the top or bottom row of the computation matrix and goes row by row, performing the necessary calculations.

Learning on the GPU

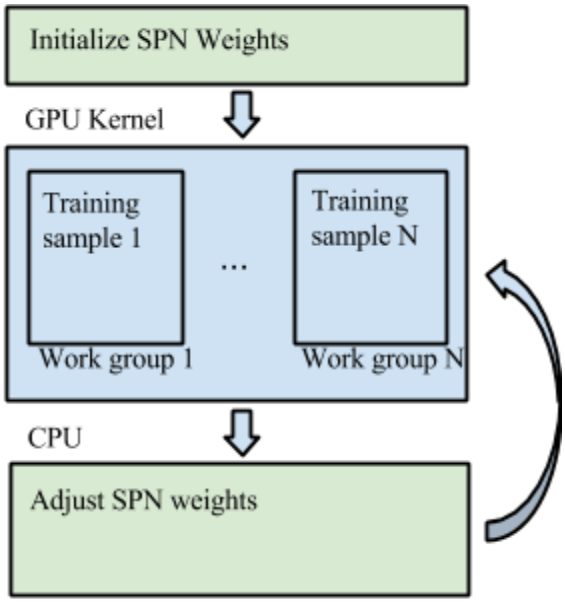

Figure 4: Process of learning SPN weights using the GPU through offline batch learning.

Using a GPU to do the computation for learning can drastically speed up the process. We can apply the expectation-maximization algorithm to learn the weights of the SPN. As mentioned previously, OpenCL only allows synchronization between work items in the same work group. That means computation in different work groups need to be independent of each other. We can parallelize the learning process by using offline batch learning.

When training the network with batch learning, the weights of the SPN are not updated after calculating the error for a single training sample. The weights are only updated once the error for all the training samples are calculated. We can take advantage of this by having each work group compute the errors for each training sample. Once the errors for all the training set is completed, the results are sent back to the host CPU, the weights of the SPN are then adjusted and the task to

compute the new errors are dispatched to the GPU. By doing this, each GPU work group can perform the expectation maximization algorithm on its own training sample, and work independently from other work groups, which allows the GPU to fully utilize its parallel nature.

Testing the SPN

To test the implementation for the SPN learning algorithm for both the CPU and GPU, a SPN was made to perform image completion. This task was described in the original paper for SPNs, where Poon and Domingas achieved great results in completing pictures of faces. In their experiment, they were using a cluster of very powerful computers to train the sum product network. For this project, a similar, but much smaller in scale, problem was used. The data set used was a modified version of the notMNIST dataset, which had a collection of images of the letter A.

In this scenario, the sum product network was trained with samples of 16x16 black and white images of the letter A. Images were taken from the notMNIST dataset and were preprocessed to set each of the pixels to either black or white. This is so the domain of the variable nodes is in {0,1}. The variables of an SPN can be extended to be continuous, but this was outside the scope of this project.

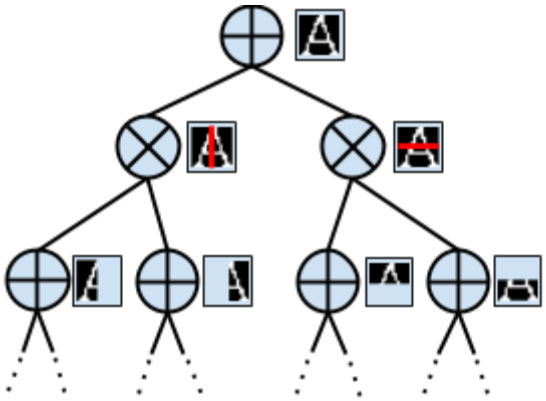

Figure 5: SPN architecture for image completion

The architecture used, visualized in *Figure 5*, was described in the original paper for Sum Product Networks. Direct children of product nodes are different partitions of the region of the node. Direct children of sum nodes are different decompositions of a region in the image. Training the SPN will inform it about which patterns and sub-patterns occur most often with one another.

Once the SPN is trained, the SPN can perform image completion by calculating the MAP state given the right half of the image as evidence. *Figure 6* shows the results from the training set. The first row shows the original images. The second row shows the right half of the image which was given as evidence. The final row shows the image which was completed by the SPN. The algorithm seems to be able to complete the left side of the image reasonably well. It usually identifies whether an A is wide or thin, but it has some trouble filling in certain uniquely shaped As.

Figure 6: Results from testing the trained SPN.

Results

The CPU program was used to product the results above. The OpenCL implementation has several challenges. For example, the language used for the OpenCL kernel program is based in C99, which is a previous version of the C language specification. It makes implementation significantly more difficult because it doesn't have many features of higher level languages. Additionally, one major drawback of using average consumer GPUs for computation is that many GPUs only support single floating point calculations. Single floating point only has approximately 7 significant digits (base 10) of precision and the smallest positive number is around 10^{-38}. Often times, this is not nearly enough for training the weights of an SPN. As a result, the OpenCL implementation could not produce the same results as the CPU program. This is a problem that is very difficult to solve without hardware which supports double precision.

Conclusion

The main motivation for an OpenCL implementation is that it can be used to accelerate SPN learning on regular consumer graphics cards. However, since average consumer GPUs generally do not have double precision calculations, they are unable to properly train a SPN using the method proposed in this project. Recently released high-end consumer graphics cards are starting to support double-precision calculations. This method may be more applicable in a few more years when double-precision GPU support is ubiquitous.