# Implementation of Fast Artificial Neural Network for Pattern Classification on Heterogeneous System

Devendra Singh Bharangar, Prof.Amit Doeger, Prof.Yogesh Kumar Mittal

**ABSTRACT--** Neural networks have been part of an attempt to emulate the learning curve of the human nervous system. Graphics Processing Units (GPUs) that come with a Graphics card have hundreds of processing cores, and have highly parallel architecture. Because of the highly parallel architecture of GPUs, it suits very well for parallel architecture such as Neural Network. In fact, GPUs have become a General Purpose Processor, and is a good option for implementation of many Parallel Algorithms, including ANN. Further recent advancements in GPU computing have made it easier to utilize the resources of a GPU. Specifically the programming model has been made much simpler. NVIDIA OR AMD OpenCL for example can be extremely helpful in accelerating ANN algorithms on GPUs. Thus OpenCL accelerated ANN algorithms can be used in many real-time applications, including image processing, object classifications, voice recognition and in a number of systems which require intelligence and auto control. This research thus aims at implementation of a Neural Network on a GPU in order to improve the performance as compared to CPU implementation in a particular application. Particularly, we have chosen NVIDIA OR AMD's GPU and OpenCL platform for this implementation.

***Keywords-*** **GPUs, *ANN Classifier,Training*, NVIDIA OR AMD's CUDA/OpenCL**

## 1. INTRODUCTION

GPUs have become a General Purpose Processor, and are a good option for implementation of many parallel algorithms. In addition to this the GPU-based ANN is much more cost effective as compared to FPGA or ASWIC based solutions. Thus implementation of an Artificial Neural Network on a GPU provides improved performance as compared to CPU implementation.

Life before NVIDIA's Compute Unified Device Architecture (CUDA) [1] was extremely difficult for the programmer, since the programmers needed to call graphics API (Open GL, Open CV etc.). This also has a very slow learning rate. CUDA solved all these problems by providing a hardware abstraction, hiding the inner details of the GPUs, and the programmer is freed from the burden of learning graphics programming. CUDA can be extremely helpful in accelerating ANN algorithms on GPUs [2] [3]. Then the other widely popular standard is OpenCL, which we used in this work.

OpenCl/CUDA is C language with some extensions for processing on GPUs. The user writes a C code, while the compiler bifurcate the code into two portions. One portion is delivered to CPU (because CPU is best for such tasks) while the other portion, involving extensive calculations, is delivered to the GPU(s) that executes the code in parallel. With such a parallel architecture, GPUs [4] provide excellent computational platform, not only for graphical applications but any application where we have significant data parallelism. Pattern Recognition is defined as the process whereby a received pattern/signal is assigned to one of a prescribed number of classes. The terms pattern means something that is set up as an ideal to be imitated. Humans are good at pattern recognition. The Pattern Recognition means identification of the ideal object. Recognition should be preceded by development of the concepts of the ideal or model or prototype. This process is called Learning or Training.

Thus CUDA/OpenCL accelerated ANN algorithms can be used in many real-time applications, including image processing, object classifications, voice recognition and in a number of systems which require intelligence and auto control. Thus implementation of an Artificial Neural Network on a GPU improves the performance as compared to CPU implementation. Such a scheme applied on Convolutional Neural Networks can be used in future researches on implementation of pattern matching on GPUs with much more accuracy and speed.

In Part II, we discuss details about the GPU, its architecture, and how we can utilize these commodity GPUs as general computational device.

In part III, we discuss how we implemented a general ANN on GPU, while in at the end we have summarized various results along with conclusion and scope for future scope.

## 2. ARCHITECTURE OF MODERN GRAPHICS CARD

GPU computing is the use of a GPU (graphics processing unit) to do general purpose scientific and engineering computing. The model for GPU computing is to use a CPU and GPU together in a heterogeneous computing model. The sequential part of the application runs on the CPU and the computationally-intensive part runs on the GPU [11]. From the user's perspective, the application just runs faster because it is using the high-performance of the GPU to boost performance. Fig 1 shows a typical Graphics card layout.

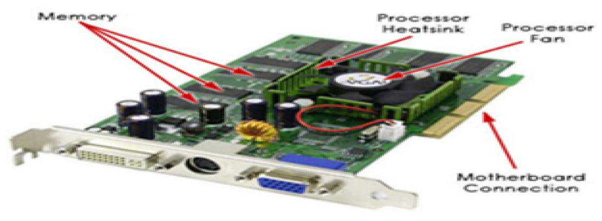

Fig 1: Example of a Typical GPU card

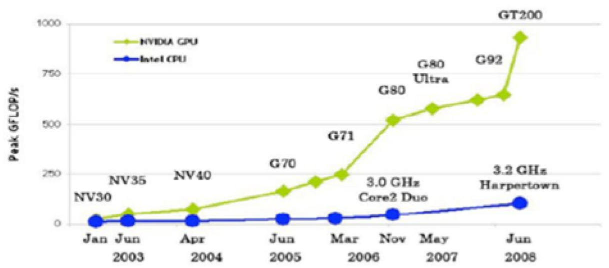

Fig 2: Comparison of Computational Power (in GFLOPs, Giga Floating Point Operations per second) of a CPU and GPU

The application developers have to modify their applications to take the compute-intensive kernels and map them to the GPU. The rest of the application remains on the CPU. Mapping a function to the GPU involves rewriting the function to expose the parallelism in the function and adding “C” keywords to move data to and from the GPU. GPU computing is enabled by the massively parallel architecture of NVIDIA’s CUDA architecture and now OpenCL. The OpenCl architecture consists of 100s of processor cores that operate together to crunch through the data set in the application.

A GPU (Graphics Processing Unit) is a processor attached to a graphics card dedicated to calculate floating point operations. A graphics accelerator incorporates custom microchips which contain special mathematical operations commonly used in graphics rendering. The efficiency of the microchips therefore determines the effectiveness of the graphics accelerator. They are mainly used for playing 3D games or high-end 3D rendering. A GPU implements a number of graphics primitive operations in a way that makes running them much faster than drawing directly to the screen with the host CPU. The most common operations for early 2D computer graphics include the Bit BLT operation, combining several bitmap patterns using a Raster Op, usually in special hardware called a "blitter", and operations for drawing rectangles, triangles, circles, and arcs. Modern GPUs also have support for 3D computer graphics, and typically include digital video–related functions.

A GPU implements a number of graphics primitive operations in a way that makes running them much faster than drawing directly to the screen with the host CPU. The most common operations for early 2D computer graphics include the Bit BLT operation, combining several bitmap patterns using a Raster Op, usually in special hardware called a "blitter", and operations for drawing rectangles, triangles, circles, and arcs. Modern GPUs also have support for 3D computer graphics, and typically include digital video–related functions.

A pipelined architecture is the standard procedure for processors as it breaks down a large task into smaller individual grouped tasks. When a set of instructions are transferred to the GPU, then GPU breaks up the instructions and sends the broken up instructions to other areas of the graphics card specifically designed for decoding and completing a set of instructions. These pathways are called stages. The more stages the graphics card has, the faster it can process information as the information can be broken down into smaller pieces while many stages work on a difficult instruction.

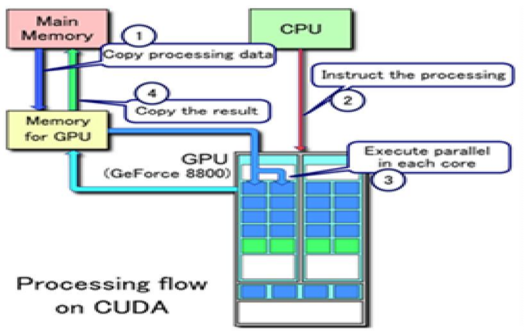

Figure 3: Processing flow on CUDA/OpenCL

Simple CUDA & OpenCL programs have a basic flow:

1. The host initializes an array with data.
2. The array is copied from the host to the memory on the CUDA/OpenCL device.
3. The CUDA/ OpenCL device operates on the data in the array.
4. The array is copied back to the host.

## 3. ANN ON GRAPHICS CARD

### 3.1. Motivation

Any particular layer in ANN has a number of processing nodes or Neurons. These Neurons work independently. That means each of these processors can work independently. Therefore, we intend to do this processing in parallel on having 100s of processor cores. In this work we utilized OpenCl parallel programming platform on a Compatible GPU. Here we will explore how a general Neural Network can be implemented on a GPU. We will then compare the performance of GPU based-neural network with a CPU implementation.

To develop an Artificial Neural Network on a Graphical Processing Unit, we used OpenCL for its implementation. The main idea is to do the ANN calculations faster and thus making it suitable for real-time applications. GPUs have hundreds of processing units and have a highly parallel architecture that clearly maps to ANN as ANN is also a massively parallel system. In addition to this the GPU-based ANN is much more cost effective as compared to FPGA or ASWIC based solutions.

This research aims at implementation of an Artificial Neural Network on a GPU in order to improve the performance as compared to CPU implementation.

Thus OpenCl accelerated ANN algorithms can be used in many real-time applications, including Image processing, character recognition, object classifications, voice recognition and in a number of systems which require intelligence and auto control.

In this direction, we proposed following architecture for character recognition on GPU. We developed a digital circuit and then converted it into ANN. We first performed certain experiments before implementation. We started with a two layered neural network with one output node. In this there are only two layers (i.e. input and output) and the hidden layer is not there. The layers of nodes between the input and output layers, which is not seen by outside, are called hidden layers. In the nets, weighted sums of the inputs are calculated by the output node. The output nodes then compare the weights.

Assume that a linear decision boundary will be used to classify samples into two classes and each sample has m features. If the discriminants function has the form

$$D=w0+w1x1+\ldots+wm.xm, \qquad (1)$$

Then D=0 is the equation (1) of the decision boundary between two classes. The weights w0, w1,..., wm are to be chosen to provide good performance on the training set A with feature vector x=(x1, x2,..., xm) is classified into one class, say class1 if D > 0 and into other class, say class -1 if D < 0.If D=0 the sample x may be classified arbitrarily, here we are assigning it to class 1. We then considered a problem of calculating logical AND of two inputs using ANN. In logical functions, 1 represents “true” or presence of binary feature, while 0 represents “false” or absence of a binary feature. If x1 and x2 are binary features,x1=1 and x2=1 means that both features are present ,so x1 AND x2=1.However if one of the feature is absent (x1,x2,or both are 0) then x1 AND x2 = 0.One of the possible lines that separates the point (1,1) from(0,0),(0,1)and(1,0) has the equation -1.5+x1+x2=0(see figure).Thus a set of weights for the two layer net that produces a logical AND of its inputs is w0=-1.5,w1=1 and w2=1.

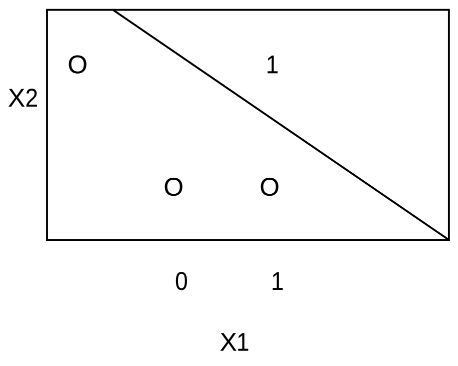

Figure 4: The separating line -1.5+x1+x2=0 for a logical AND pattern

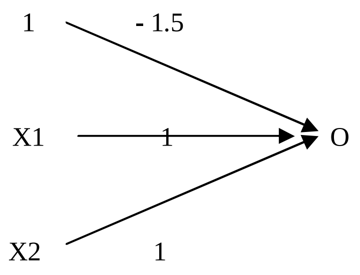

Figure 5: The two layer net implementing a logical AND

We first performed certain experiments before entire implementation as in examples below. The first example shows the AND implementation on ANN.

Thus the result implies that the output =1 only when both inputs are 1 else output =0 as is the function of AND gate.

In second example, we have increased number of inputs and thus modified C will be as shown below. The time required for this performance on C is to be noted.

Thus we have done few examples of many inputs and one output. Similarly we did examples on two inputs and two outputs & also many inputs & two outputs. Our main aim here is to show that a GPU can be the best option for an ANN implementation especially in case of pattern recognition.

Since we have considered an example to classify characters A and B else unrecognized ones. Its C implementation is as shown below and its corresponding OpenCL implementation is also shown below the time required for its execution is noted. Here we considered a character with fixed number of bits i.e. n=15 that corresponds to the character

i.e. A -0 0 1 0 0 0 1 1 1 0 1 0 0 0 1.

B - 0 1 1 1 0 0 1 1 1 0 0 1 1 1 0.

Bit map of A,

00100

01110

10001

Bit map of B,

01110

01110

01110

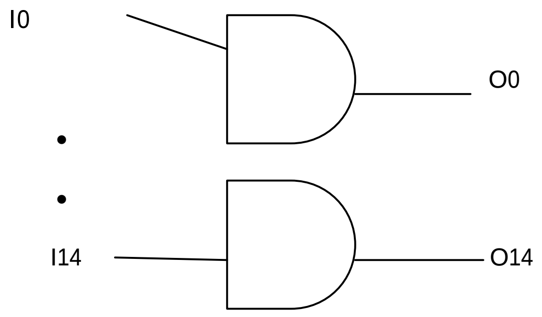

Figure 6: Digital circuit with many inputs and two outputs

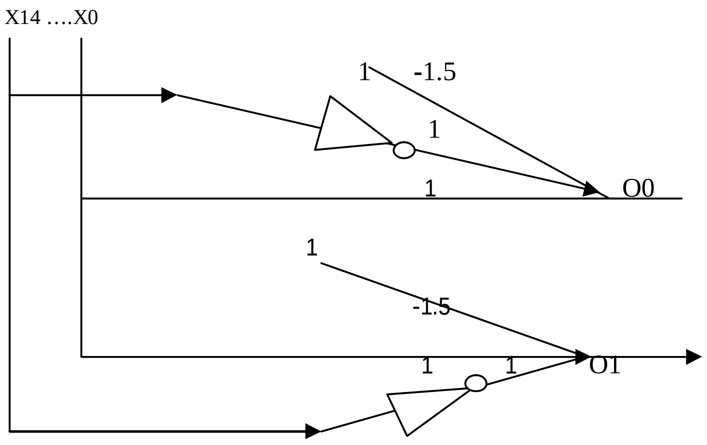

Figure7.An ANN implementation of the above digital circuit

| O0 | O1 | output |
|---|---|---|
| 1 | 0 | A |
| 0 | 1 | B |
| 1 | 1 | unrecognized character |
| 0 | 0 | |

We have designed a digital circuit with many inputs and two output (x1 & x2) such that we obtain the combination of 4 output patterns. If the output combination is x1=x2=0 or 1, then the character is unrecognized else if x1=1, x2=0 then character is A and if x1=0, x2=1 then the character is B.This digital circuit is converted into its equivalent neural network that will implement the equation- S1= ΣXA, S2= ΣXB, B=N-0.5,here we assumed no of bits n+15,Y1=S1-B,and Y2=S2-B.

Here we'll do OpenCL implementation of ANN and also note down the time required for this performance.

### 3.2 Experiments and Analysis

**We observed that the output of the neural network can be defined as Y= ΣXi – B.**Now our main focus is to show the performance of **neuro- reduction i.e. ΣXi.**

It is just a sum reduction process. Here is where the actual speed up is obtained when it is implemented on GPU as compared to CPU.This process will take less time on GPU since it has parallel architecture. Suppose that i=8 (i.e. 01100111), the CPU implementation for this case will take 8 iterations as the addition is sequential. But in case of GPU, it will end up in 3 iterations only i.e. log2 8=3.Here the bit addition takes place in parallel.

Performance speedup = iteration taken by CPU vs. iterations required by GPU = 8/3

= 2.6

Thus GPU is 2.6 times (260%) faster than CPU.

Now if we increase the number of inputs to 1024, then CPU will require 1024 iterations while GPU will requires $\log_2$ 1024=10 iterations. Hence the performance speed up

= 1024/10
= 102.4

Theoretical speed up = 102.4

Thus GPU is 102.4 times faster than CPU.

### 3.3 Important Results

To prove the computational power of a GPU, we did large no. of experiments with and without shared memory. The results have been summarized in to tabular form for better understanding.

TABLE1.

REDUCTION FULLY OPTIMIZED

| N | GRID_SIZE | BLOCK_SIZE | LOAD PER THREAD | TIME TAKEN |
|---|---|---|---|---|
| 2^22 | 64 | 128 | 512 | 1.7805 |
| 2^22 | 64 | 256 | 256 | 1.6629 |
| 2^22 | 64 | 512 | 128 | 1.636 |
| | | | | |
| 2^22 | 128 | 128 | 256 | 1.6553 |
| 2^22 | 128 | 256 | 128 | 1.6026 |
| 2^22 | 128 | 512 | 64 | 1.6545 |
| | | | | |
| 2^22 | 256 | 128 | 128 | 1.6 |
| 2^22 | 256 | 256 | 64 | 1.6126 |
| 2^22 | 256 | 512 | 32 | 1.7527 |
| | | | | |
| 2^22 | 512 | 128 | 64 | 1.5948 |
| 2^22 | 512 | 256 | 32 | 1.6992 |
| 2^22 | 512 | 512 | 16 | 1.9711 |

## 4. CONCLUSION AND FUTURE SCOPE

In this paper, the discussion is all about our work for the implementation of an Artificial Neural Network on a GPU, which optimally will improve the performance as compared to CPU implementation. Such a scheme applied on Convolutional Neural Networks can be used in future researches on implementation of pattern matching on GPUs with much more accuracy and speed. OpenCL accelerated ANN algorithms can also be used in several real-time applications, including image processing, voice recognition and in a number of systems which require intelligence and auto control including object classifications.

In future work with same purpose, OpenCL accelerated algorithms might be useful to have solution for the problem of pattern classification on other types of processors such as DSP, FPGA etc. with respect to CPU.

Although, we have worked suited for heterogeneous system having a CPU and a GPU, however, the same can be tested on the embedded systems, if required tools are available. For example we can develop such a system on a Nokia phone. We might have to wait because at present mobile phone manufacturers have not come up with tools for porting OpenCL on mobile phones.

The application of mobile phone will provide portability to the users and of course, it can be used as a basic algorithm in a number of different mobile applications right from gaming to advance financial applications.

## ACKNOWLEDGEMENT

I am grateful to my guides Prof.Amit Doegar and Prof.Yogesh Kumar Mittal for his constant patient support and useful suggestions. Any suggestions to further improvement of this topic are most welcome.

## REFERENCES

[1] R. Tom, Halfhill, “Parallel processing with CUDA”, Microprocessor report. January 2008

[2] Z. Luo, H. Liu and X. Wu, “Artificial Neural Network Computation on Graphic Process Unit”, Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference, vol.1, pp. 622 – 626, 31 July-4 Aug. 2005

[3] R. D. Prabhu, “GNeuron: Parallel Neural Networks with GPU”, Posters, International Conference on High Performance Computing (HiPC) 2007, December 2006

[4] Shuai Che , Michael Boyer, Jiayuan Meng, David Tarjan , Jeremy W. Sheaffer, Kevin Skadron, “A Performance Study of General-Purpose Applications on Graphics Processors Using CUDA”, Journal of Parallel and Distributed Computing, ACM Portal, Volume 68 ,Issue 10, pp. 1370-1380 October 2008

[5] Elizabeth, Thomas, “GPU GEMS: Chapter 35 Fast Virus Signature Matching on the GPU”, Addison Wesley

[6] Pat Hanrahan, “Why are Graphics Systems so Fast?” PACT Keynote, Pervasive Parallelism Laboratory, Stanford University, September 14, 2009

[7] Kayvon Fathalian and Mike Houston, “A closer look at GPUs”, Communications of the ACM, Vol. 51 no. 10, October 2008

[8] Trendy R. Hagen, Jon M. Hjelmervik, Tor Dokken, “The GPU as a high performance computational resource”, Proceedings of the 21st spring conference on Computer graphics, pp. 21 – 26, 2005

[9] R. H. Luke, D. T. Anderson, J. M. Keller S. Coupland, “Fuzzy Logic-Based Image Processing Using Graphics Processor Units”, IFSA-EUSFLAT 2009

[10] Victor Podlozhnyuk, Mark Harris, “NVIDIA CUDA Implementation of Monte Carlo Option Pricing”, Tutorial in NVIDIA CUDA SDK 2.3

[11] “CUDA Programming Guide 3.0”, Published by NVIDIA Corporations

[12] David B. Kirk and Wen-mei W. Hwu, Programming Massively Parallel Processors: A Hands-on Approach by Morgan Kaufmann (February 5, 2010), ISBN 0123814723

[13] K. Oh, “GPU implementation of neural networks” Pattern Recognition, pp. 1311-1314. Vol. 37, No. 6. June 2004.

[14] Simon Haykin, “Neural Networks: A comprehensive foundation”, 2nd Edition, Prentice Hall, 1998

[15] Araokar, Shashank, “Visual Character Recognition using Artificial Neural Networks” Cog Prints, May 2005

[16] EarlGose, Steve Jost, Richard Johosonbaugh,”Pattern Recognition and Image Analysis”, Eastern Economy Edition, Prentice Hall, 1999

AUTHORS

- *Devendra Singh Bharangar,Assistant Professor, Aligarh College of Engineering & Technology,Aligarh,India E-mail: devendra23@hotmail.com*
- *Prof.Amit Doeger,CSE Department, National Institute of Technical Teachers Training & Research, Chandigarh, India E- mail: amit@nitttrchd.ac.in*
- *Prof.Yogesh Kumar Mittal,HOD(I.T.),Ajay Kumar Garg Engineering College, Ghaziabad,India,E-mail:ykmittal@hotmail.co*