Accelerating Sparse Deep Neural Network using GPUs

1st Satish Kumar Oraon
12041320
Indian Institute of Technology
Bhilai, India
satishkumar@iitbhilai.ac.in

2nd
MD Arsad
12040880
Indian Institute of Technology
Bhilai, India
mdarsad@iitbhilai.ac.in

3rd Raunak Kumar
12041190
Indian Institute of Technology
Bhilai, India
raunakkumar@iitbhilai.ac.in

***Abstract*—Deep Neural Networks are at the heart of Machine Learning and artificial intelligence. One cannot imagine the impact it is having is having in the field of playing with large data sets to gain wonderful results out of it. Past researches have shown that sparse Deep Neural Networks perform as better as dense DNNs with much better memory optimization and less energy consumption . The larger the size of the data sets , the better and efficient the result. The present computer hardware and architecture makes it difficult to play with large data-sets owing to memory and energy utilization constraints. This paper addresses the 2019 Sparse Deep Neural Network Graph Challenge with an implementation of this challenge using GPU programming model.Past solutions to this problem statement have used GPU as a hardware tool to implement it.Earlier GPU implementations are 4.3 times as fast as CPU implementation.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

In this era of time, the availability of large data sets and high computing powers to process it have enabled the humankind to solve complex problems such as speech recognition , artificial intelligence , machine learning , natural language processing and many such complex problems .The DNN used to process the large data sets have many more complex elements which make it difficult to store and process in the normal computers. One has to keep in mind the cost and energy usage.

This motivates the the scientific community to find new ways to process such large and dense data sets to sparse data sets which can be processed at a higher speed and much less energy consumption. It requires modifying the large data sets .Thus an emerging area of focus is to prune the network by removing network connections with small weights, making the networks sparse and thus able to either achieve similar accuracy and better performance with less memory or superior accuracy and similar performance with the same amount of memory as dense networks.The values which are not of much significance are set to 0 in successive processing of the network .

MIT has been organizing various challenges in the field of graph computation over the years . The new challenge requires inference computation on a large data-set pf neural networks . **Inference computation** means ability of a trained system to make prediction out of the given data sets . The problem details are as listed under

Input Format Y_0 It is a sparse matrix of MINST image.
W_0 It is a sparse weight matrix of the
b It is a list of bias vectors used to make data uniform .
L It is number of total layers.
Output format
1 . We will evaluate DNN for each layer ny the transformatio $Yl + 1 = h(Yl * Wl + b)$
2. The final matrix we will produce is represented by Yl
3 . The final matrix will be multiplied with W_l to get Yl+1 .
4 The $Yl+1$ will be again calculated by the operation $Yl+1+b$ where b is the bias vector
5 The Yl+1 will be fed to **REL() fuction** .
6 We will identify the non zero elements in the Yl+1 and verify it with the given truth table.

Now this requires huge computation power and memory resources as there will be a lot of memory read and write .

It is no doubt that GPU offers high computation power in addition to offering parallel computation and high throughput . Thus it becomes an obvious tool to process and compute large data sets.Thus we will use various functionalities of GPU to solve the problem

II. Approach and Design Problem

Since We need to perform matrix matrix multiplication in this problem , we need to design our matrix matrix multiplication kernel in such a way that we extract maximum parallelism to cut out the time required in matrix matrix multiplication. The maximum dimension possible of Y is 60000 * 65536 and the maximum dimension possible of W is 65536* 65536. There is two approach that can be taken to store such a large data set in the memory. We can represent both Y and W in csr format in row major . This will not only reduce the storage space but will make their access coalesced

Implementation on GPU.We assign each thread to load the non zero elements of the Y Matrix which will produce single output element of Yl+1 .The same thread will read the columns of the W matrix . This will be done so as to make the memory access coalesced which will boost up the performance . Note that the matrix W will be stored in the form of row major

. New element calculation will be done with the help of corresponding column in Yl . At the same time , the assigned thread will calculate the sum with its corresponding bias value . After that it will be passed via RelU function which will also run parallely . Once one iteration is complete , the non zero elements of Y will be calculated and stored for next iteration . This will cause a lot of memory read and write operation . This will be reduced by storing some elements in the the shared memory so that the threads in a block can access it much efficiently and quickly.

III. Baseline implementation algorithm

We are computing simple matrix matrix multiplication in our baseline implementation. Since the 60000*1024 feature matrix and 60000*1024 feature matrix can be easily fitted into the GPU memory, we are able to get valid results with it.

```
__global__ void Kernel(float *M, float*N,
    float*P)
{

    int Row=blockIdx.y*blockDim.y+threadIdx.y;
    int Col=blockIdx.x*blockDim.x+threadIdx.x;

    if(Row<60000 && Col < 1024)
    {
        float product=0;
        for(int k=0;k<1024;k++)
            product+=M[Row*1024+k]*N[k*1024+Col];
        product = product - 0.3;
        if((product)<0)
        {
            product = 0;
        }
        if((product) >32)
        {
            product = 32;
        }
        P[Row*1024+Col]=product;
    }
    __syncthreads();
}
```

60000*1024 image dataset can easily fit into the GPU memory.So it is sent as an adjacency matrix of dimension 60000*1024 and 60000 images are parallely multiplied with weight matrix which is of dimension 1024*1024 which is too stored in the simple adjacency format in our baseline implementation.In this implemenation , we are using grid whose dimension is 32 *1875 . The dimension of one block is kept to 32 * 32 . Hence 32*32 covers 1024 columns in the y direction and 1875 * 32 in the y direction.Hence simple matrix multiplication gives valid result for 60000*1024 and 60000*4096(low layers).

IV. Memory optimization using CSR representation of the weight matrix

```
__global__ void Kernel(float * IMG,int* Rptr,int*
    int id=threadIdx.x;
    int ind=blockIdx.x;
      float prdt=0;
      for(int i=Rptr[id]; i<Rptr[id+1]; i++){
          prdt+=IMG[ind*1024+Cptr[i]]*value[i];
      }
      prdt-=0.3;
      if(prdt<0)prdt=0;
      if(prdt>32)prdt=32;
     D_P[ind*1024+id]=prdt;
      __syncthreads();

}
```

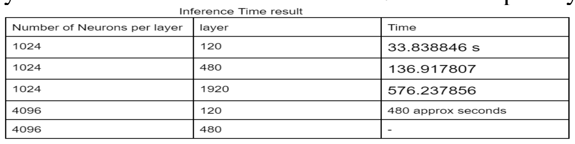
We have tried to optimize our baseline implementation by storing the weight matrix int the CSR format.

V. Evaluation Methodology

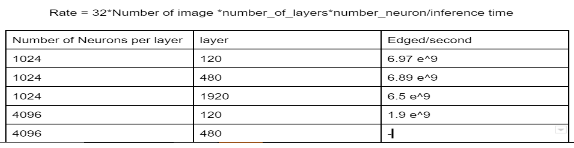
For this task, we will use google colab and its GPU . 1) GPU Tesla T4 2) number of SM 72 3) Shared memory in each SM 48 KB 4)Max thread in one block 1024
The data sets for the task will be taken from MINST.The results obtained will be calculated with the existing submissions available in the public domain.

VI. Results

On performing our test on GPU by google colab, we find out that total of 1812 images are active after 120 ,480,1920 layers for both 1024 neurons and 4096 neuron per layer .

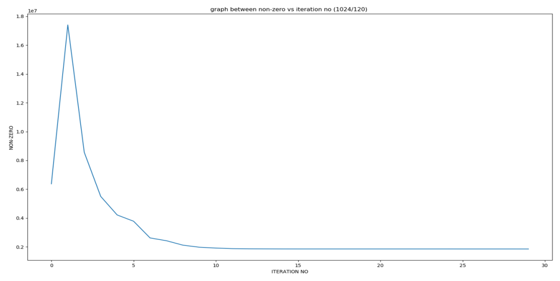

Inference Time result

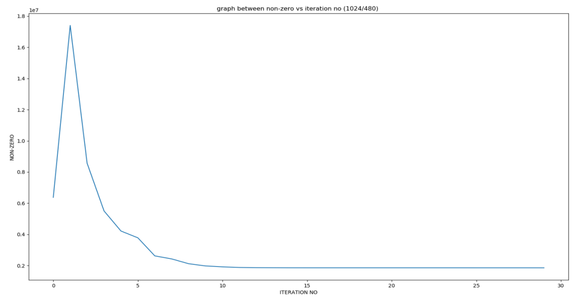
Number of Neurons per layer	layer	Time
1024	120	33.838846 s
1024	480	136.917807
1024	1920	576.237856
4096	120	480 approx seconds
4096	480	-

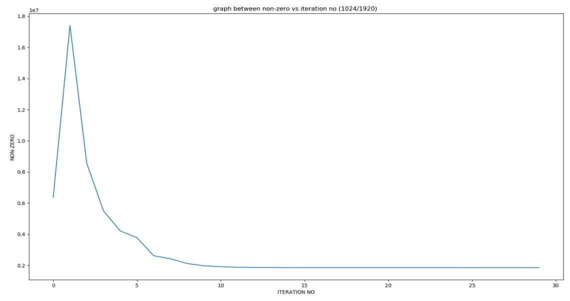

Rate = 32*Number of image *number_of_layers*number_neuron/inference time

Number of Neurons per layer	layer	Edged/second
1024	120	6.97 e^9
1024	480	6.89 e^9
1024	1920	6.5 e^9
4096	120	1.9 e^9
4096	480	-|

As it is clear from the above result table , the rate of inference edges decrease as we move to larger neurons per layer.
The number of non zeroes become constant after few iterations.The number of full rows in feature matrix increases so it was decided not to store feature matrix in CSR format The corresponding graph is attached for reference.







As it can be inferenced from the graph , the number of non zeros are becoming constant after 28th iteration.

Kernel Call statistics 1024 /120 Layer in baseline implementation There are total 1812 active images after 120 layers.Each kernel call is taking 0.166812 seconds . This time is consumed in matrix matrix multiplication of 60000*1024 feature matrix with 1024*1024.The time excludes time of copying feature matrix and weight matrix from device to host and vice versa.Hence the matrix matrix multiplication is taking 20.01744 seconds for 120 layers.

VII. Conclusion

Deep neural networks have significantly advanced the state-of-the-art across a number of domains, revolutionising the field of machine learning. However, the hardware required to implement deep neural network topologies is becoming increasingly taxed by their size. The sparsification of these neural networks has been the subject of extensive research over the past ten years in an effort to reduce storage and runtime costs.

VIII. Acknowledgement

This project would not have been possible without the guidance of our professor Dr Vishwesh Jatala @IIT Bhilai

IX. References

1. X. Wang, Z. Lin, C. Yang and J. D. Owens, "Accelerating DNN Inference with GraphBLAS and the GPU," 2019 IEEE High Performance Extreme Computing Conference (HPEC), 2019, pp. 1-6, doi: 10.1109/HPEC.2019.8916498.

2 . J. Wang et al., "Performance of Training Sparse Deep Neural Networks on GPUs," 2019 IEEE High Performance Extreme Computing Conference (HPEC), 2019, pp. 1-5, doi: 10.1109/HPEC.2019.8916506.

3 . M. Bisson and M. Fatica, "A GPU Implementation of the Sparse Deep Neural Network Graph Challenge," 2019 IEEE High Performance Extreme Computing Conference (HPEC), 2019, pp. 1-8, doi: 10.1109/HPEC.2019.8916223.