Parallel Backpropagation Neural Network Training Techniques using Graphics Processing Unit

Muhammad Arslan Amin[1], Muhammad Kashif Hanif[2],
Muhammad Umer Sarwar[3], Abdur Rehman[4],
Fiaz Waheed[5], Haseeb Rehman[6]
Department of Computer Science,
Government College University,
Faisalabad, Pakistan

***Abstract*—Training of artificial neural network using backpropagation is a computational expensive process in machine learning. Parallelization of neural networks using Graphics Processing Unit (GPU) can help to reduce the time to perform computations. GPU uses a Single Instruction Multiple Data (SIMD) architecture to perform high speed computing. The use of GPU shows remarkable performance gain when compared to CPU. This work discusses different parallel techniques for the backpropagation algorithm using GPU. Most of the techniques perform comparative analysis between CPU and GPU.**

***Keywords*—Artificial neural network; backpropagation; SIMD; CPU; GPU; machine learning**

I. Introduction

An Artificial Neural Network (ANN) [1] is created initially inspired by the functionality of human brain where a large number of neurons are interconnected to process information. ANN plays a vital role to analyze large scale of data. There exist various algorithms of ANN which can be utilized in a vast variety of fields. ANN is mostly used for pattern recognition and classification [2]. Backpropagation [3] is an algorithm of ANN that is mostly used due to its efficiency and simple implementation. Training and testing of ANN is a time consuming process which requires a large computational cost. There is need to increase the speed of training, testing and reduce the computational cost [4]. Parallel computing can help to increase the speed and reduce the cost of computation to train and test ANN.

Backpropagation neural network consists of single input, output, and one or more hidden layers. The neurons in the same layer are independent. The appropriate weights among neurons are obtained by performing multiple iterations. Backpropagation has forward and backward pass. In forward pass, the input vector of each layer is computed in each iteration. While, in backward pass, calculation of gradient descent and updation of weights is performed.

GPU consists of a large number of cores for parallel execution and performance enhancement of different applications [5]. GPU have already been used to solve computational complex problems in different areas like physics simulations, molecular dynamics, and scientific computing [6]. The programming model for NIVIDIA graphics card is *Compute Unified Device Architecture* (CUDA). CUDA programming model provides shared memories, a hierarchy of thread groups, and barrier synchronization to accelerate the applications [6]. A CUDA kernel can execute large number of threads concurrently. GPU can also work with neural networks in order to perform their operations and obtain efficient results. In this study, we have reviewed the implementation of backpropagation algorithm techniques using GPU. The execution and training of ANN on GPU can be performed in different steps, i.e., data preparation, transfer of data from CPU to GPU, kernel execution, and then results are transferred to the host [7].

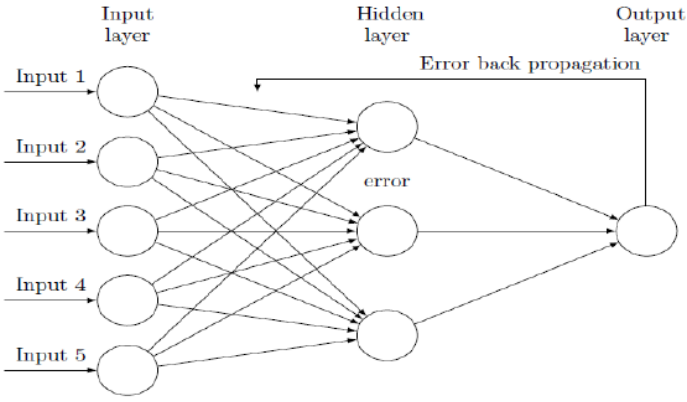

Fig. 1. Backpropagation Neural Network [7].

The rest of this paper is organized in different sections. Section II describes the serial backpropagation algorithm. Section III describes the parallel techniques for backpropagation neural network. Finally, Section IV presents the conclusion.

II. Backpropagation Algorithm

There exist various algorithms to train an ANN. Backpropagation is one of the popular algorithm which is mostly used due to programming ease and has a power to manipulate large amount of data. A neural network contains an input, output, and one or more hidden layers. A layer consists of a vector of neurons and weights together with an activation function. These layers are connected with succeeding ones as shown in Fig. 1.

The number of hidden layers can be determined by the problems complexity [8]. The first part of backpropagation algorithm is feed-forward pass which presents inputs to the network and propagate forward to produce the output. In backward pass, the output is compared with the desired output.

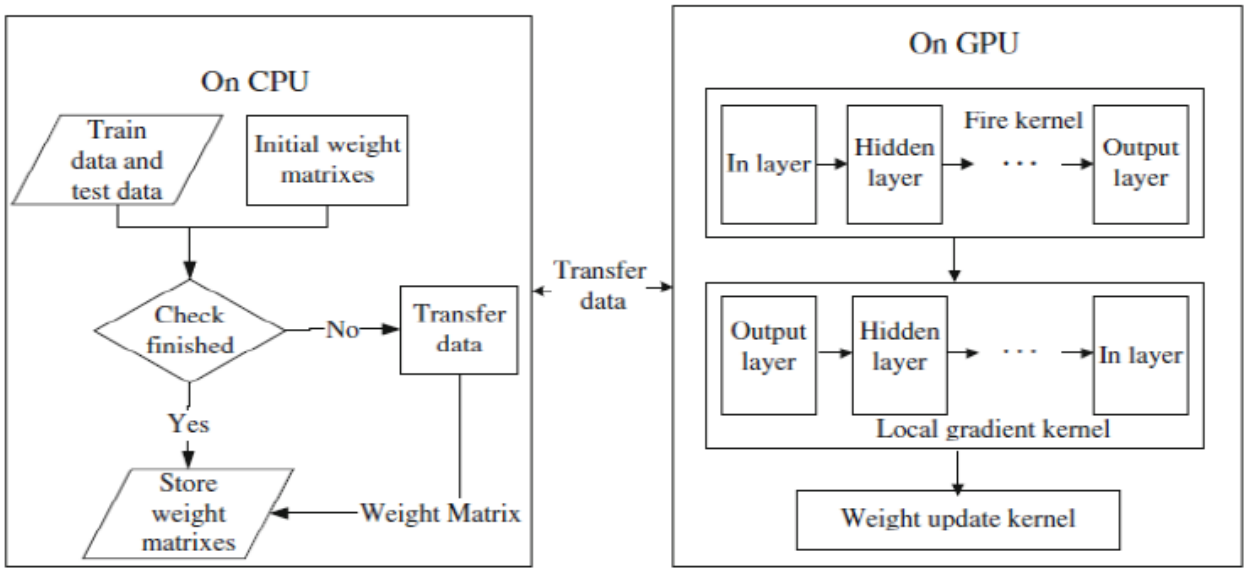

Fig. 2. Flow of BackPropagation Network on GPU [2].

The weights are updated according to error correction rule [7], [9].

III. Parallelization Backpropagation Neural Network using CUDA

Training and testing of ANN has large computational cost and time taking process. Parallelism of neural networks using GPU can help to reduce the time to perform computations. Different researcher have used parallel computing to enhance the performance of backpropagation algorithm. Backpropagation has been implemented using OpenMP [11], MPI [12] and GPU using CUDA [4], [10].

The workflow of the backpropagation algorithm using GPU is described in [5]. This can be summarized as following:

1) Read input data.
2) Random initialization of weights.
3) Copying weights to the GPU.
4) Copying input to the GPU.
5) Neural network initialization.
6) Calling feed-forward kernel (input to hidden layer).
7) Calling feed-forward kernel (hidden to output layer).
8) Call the kernel for delta calculation.
9) Kernel for updating weights (input to hidden layer).
10) Kernel for updating weights (hidden to output layer).

Backpropagation algorithm implemented by Brito et al. on the GPU contains three functions. The first function (i.e., DeltaCalculation) calculates error between output and hidden layers. This function is used in the process of updating weights in backpropagation. The function UpdateInputWeights updates weights which connects input to hidden layer. Number of threads in a block are equal to the number of inputs. the Number of nodes in hidden layer are equal to number of blocks. The UpdateHiddenWeights updates weights for hidden to output layer. In this function, number of threads are same as number of nodes in hidden layer and number of blocks are equivalent to the number of nodes in output layer. The execution of blocks and threads update the weights in parallel [5]. The typical execution flow of GPU enabled backpropagation algorithm is shown in Fig. 2.

The implementation of backpropagation algorithm in parallel can be done using vector and matrix operations. Arithmetic and vector-matrix products are considered as the types of parallel operations. Vector and matrix operations can be performed using CUBLAS. The utilization of kernel is essential unless CUBLAS do not perform all the operations. In [4], the comparative analysis on cancer and mushroom datasets using CPU and GPU are performed. This comparative study is performed by changing the size of hidden neurons in CPU and GPU. When the number of hidden neurons increases, the computation is becoming more complex due to size of sub matrices. The test results indicates the speedup of 46 and 63 times in cancer and mushroom data, respectively [4].

The implementation of backpropagation neural network in batch mode has been demonstrated in [10]. Every layer in neural network exists in the form of matrix. The matrices are distributed over multiple GPUs in order to gain high speed up. The implementation of GPU requires CUBLAS and CUDA kernel. The framework for training of backpropagation algorithm using multiple GPUs is shown in Fig. 3. Every GPU feed forwards the input data to successive one's for the calculation of gradients and training errors. The information about gradients and training errors is collected by the first GPU from all other GPUs to sum them (training errors and gradients) respectively. The gradients are moved to every GPU for updating the weights. This process continues until the goal is attained. The technique of using multiple GPUs attains higher speed up than other techniques. Multi GPU training was approximately 51.33 times faster than CPU. On the other hand, training on single GPU is just 11.99 times faster [10].

The parallel implementation of backpropagation algorithm using multicore processors and GPUs are discussed in [13].

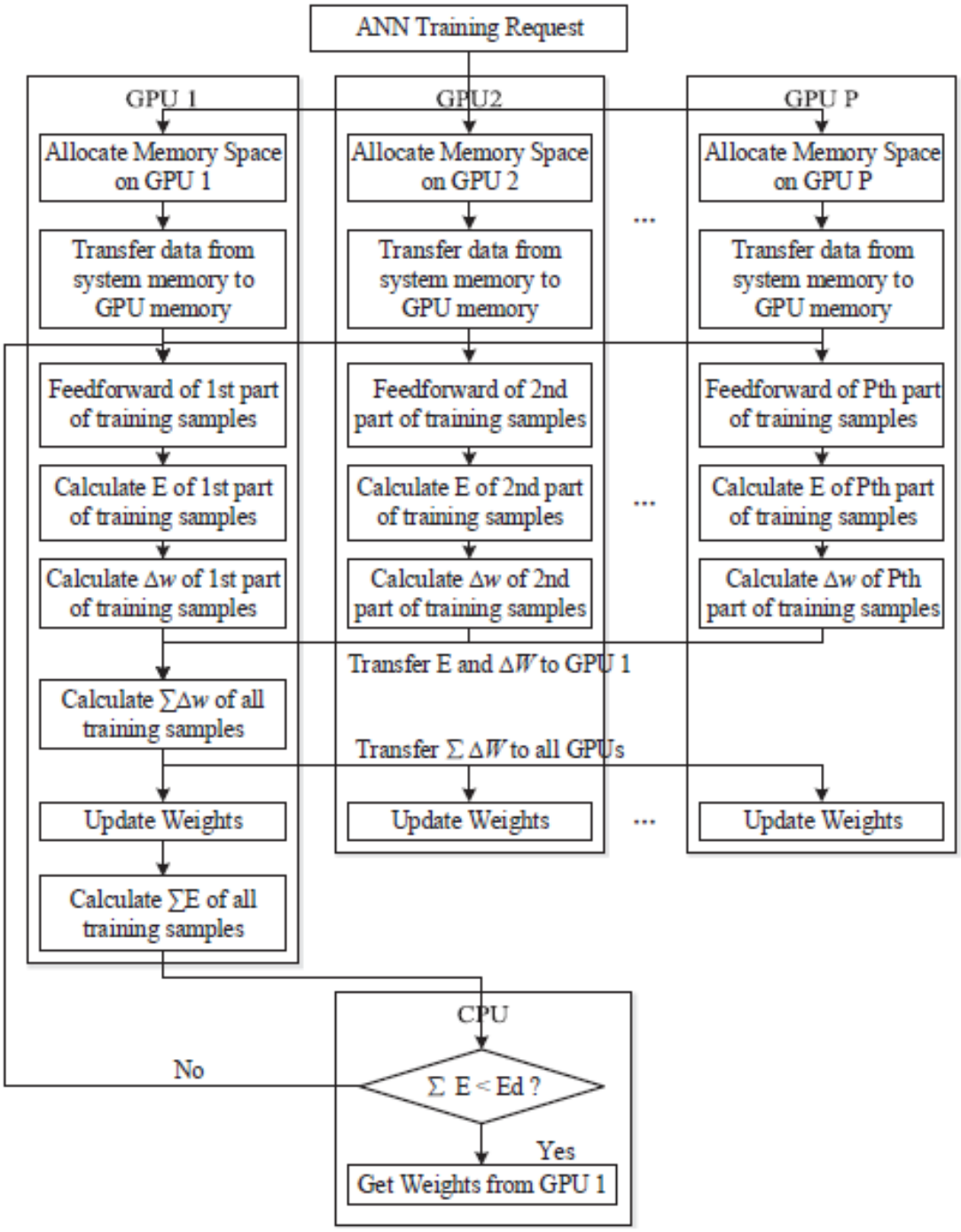

Fig. 3. Parallel backpropagation training on multiple GPUs framework [10].

They have also compared three versions of backpropagation algorithms, i.e., sequential or classical backpropagation algorithm, OpenMP shared memory multiprocessing algorithm where parallel computations are performed on multicore CPU, and GPU implementation of backpropagation algorithm [13]. The results demonstrated that the parallel executions can explore a more prominent number of solutions and attain a low mean square error. Different numbers of ANNs can be trained in parallel simultaneously. GPU implementation showed approximately 496 times more solutions when compared with OpenMP implementation [13].

IV. Conclusion

Training of ANN with backpropagation is a time taking and computational expensive process. GPU based parallelism of neural networks can help to decrease the training time. In this study, three techniques of parallel backpropagation neural network, i.e., using single GPU for training and testing, using multiple GPUs, and training many neural networks

simultaneously are discussed. The speed of neural network can be improved using GPU when compared to the CPU version. The CPU performs better for small number of attributes and the GPU version performed efficiently on a dataset with large scale of attributes.

REFERENCES

[1] A. Abraham, "Artificial neural networks," *handbook of measuring system design*, 2005.

[2] Y. Wang, P. Tang, H. An, Z. Liu, K. Wang, and Y. Zhou, "Optimization and analysis of parallel back propagation neural network on gpu using cuda," in *International Conference on Neural Information Processing*. Springer, 2015, pp. 156–163.

[3] J. C. Chaudhari, "Design of artificial back propagation neural network for drug pattern recognition," *International Journal on Computer Science and Engineering (IJCSE)*, pp. 1–6, 2010.

[4] X. Sierra-Canto, F. Madera-Ramirez, and V. Uc-Cetina, "Parallel training of a back-propagation neural network using cuda," in *Ninth International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2010, pp. 307–312.

[5] R. Brito, S. Fong, K. Cho, W. Song, R. Wong, S. Mohammed, and J. Fiaidhi, "Gpu-enabled back-propagation artificial neural network for digit recognition in parallel," *The Journal of Supercomputing*, vol. 72, no. 10, pp. 3868–3886, 2016.

[6] NVIDIA, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 2015.

[7] S. M. Wagh and D. Pawar, "Gpu parallelization of back-propagation neural network," *training*, vol. 6, no. 1, 2017.

[8] N. Murata, S. Yoshizawa, and S.-i. Amari, "Network information criterion-determining the number of hidden units for an artificial neural network model," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 865–872, 1994.

[9] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "Gpgpu processing in cuda architecture," *arXiv preprint arXiv:1202.4347*, 2012.

[10] S. Zhang, P. Gunupudi, and Q.-J. Zhang, "Parallel back-propagation neural network training technique using cuda on multiple gpus," in *IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization (NEMO)*, 2015, pp. 1–3.

[11] M. Araiijo, E. Teixeira, F. Camargo, and J. Almeida, "Parallel training for neural networks using pvm with shared memory," in *The 2003 Congress on Evolutionary Computation, 2003. CEC'03*, vol. 2. IEEE, 2003, pp. 1315–1322.

[12] R. K. Thulasiram, R. M. Rahman, and P. Thulasiraman, "Neural network training algorithms on parallel architectures for finance applications," in *Proceeding of International Conference on Parallel Processing Workshops*. IEEE, 2003, pp. 236–243.

[13] J. A. Cruz-López, V. Boyer, and D. El-Baz, "Training many neural networks in parallel via back-propagation," in *IEEE International on Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017, pp. 501–509.