**Accelerate Graph Neural Networks**

**By Parallel Computing CUDA/GPU via Optimzer**

***Mina Shahmoradi, José Luis Abellán Miguel***

UCAM Universidad Católica San Antonio

[mina.shahmoradi@gmail.com](mailto:mina.shahmoradi@gmail.com),

# Abstract

Recent deep learning models encounter a gigantic data which has been grown incredibly. At a simple glance we will be noticed the old solutions will not satisfy our nowadays demands. The outdated data structures and antiquated processors have been abolished and are out of services. Flourishing molecular biology, drug discovery, social network, business e-commerce lead us to a novel high dimensional graph-structured. Graph Neural Network (GNN) is a solution to inference valuable data from nodes which are formed by molecules or users in order to extract desirable result. GNN is applied in product recommendation, image classification, renewable energy and biology. GNN relies on Graphic Processor Unit GPU [2]. GPU has high memory bandwidth and processing potency rather than CPU, so it accelerates parallel computing. In this research Graph Convolutional Neural Network (GCNN) with optimizer algorithms have been used to enhance GPU performance while the code has been written in C++ and CUDA API. The fundamental reason for using GCNN is due to linear layers and nonlinear activation to elevate precision while Convolutional algorithm eliminates redundant data and transform train set to smaller ones which can help to speed up model training. Eventually we measure running-time with and without optimizer and figure it up that optimizer algorithm could train model 5 percent faster according to observations. These experiments illustrate us that the model with optimizer obtains significantly higher accuracy and efficiency.

# Introduction

Graph Neural Network (GNN) is one of deep learning methods which has been designed to perform inference on graph-structured data. For instance GNN is used in recommendation systems, image classification, to discover a new catalyst from chemical reactions in order to convert sun or wind energy to fuel energy. Also, it is used in pharmacy to discover a new drug from decoding bonds between molecules or atoms and even more complicated structures such as proteins and mRNA.

On the other hand processor has an important role to achieve our goals either in running time or in accuracy. GNN relies on Graphic Processor Unit GPU. GPU has high memory bandwidth and processing potency to accelerates parallel computing [2]. Moreover, the Computer Unified Device Architecture (CUDA) platform provides Application Programming Interface (API) to boost programming which utilizes GPU facilities and supports heterogeneous processing while programs can benefit from both CPU and GPU.

This research has been concentrated to augment the proper model for training arbitrary graphical data structures. Nowadays, there are huge data which has a graphical form with node and edge. The node shows the owner of information such as users or molecules while the edge presents two important factors, firstly specify that who is connected to this node and secondly how much this bond has weight.

Until now the suitable model can be Neural Network (NN) because the simple perceptron up to the complex and multiple layers would be solved by NN. But the problem is a graphical data structure which needs to have information from adjacency to extract the desirable result.

Assume we want to figure out how much a character is famous in social media and find ten top subjects which is related to them. Absolutely it is not possible by NN, and we need a Graph Neural Network to gather information and aggregate data to have a result. However, GNN is disabled to handle an arbitrary graph structured which does not have specified labeled and moves chaotic, it means that random motion crystallizes a dynamical graph.

Also, it is not a fixed image which can be modeled by Convolutional Neural Network (CNN), lets suppose that the image has two dimensions x, y while this kind of graph structured had another dimensions as z with different value towards time passing. Therefore, the image is Euclidean-structure data while finding a reasonable relationship between the node and its neighbor needs profound studies.

Using a hybrid of Graph Neural Network and Convolutional Neural Network can be the best answer to analyze arbitrary graph structured with semi labeled. Graph Convolutional Neural Network (GCNN) firstly has been introduced by Thomas N. Kipf and Max Welling in ICLR 2017.

# Related Works

#### Convolutional Networks on Graphs for Learning Molecular Fingerprints (NGF) [5]:

This is a pioneer work in Graph Convolutional which has been solved unfixed input data as arbitrary size and shape. In this research vertices indicates atoms and edges express bonds. Local filter is applied for each atom and its own neighbors, then global pooling synthesizes features from the whole atoms in the molecule.

The benefits of this way is to improve prediction accuracy. Another advantage is making affordable because fixed fingerprints takes much time and energy to encode in order to predict all possible structures. Last but not least is that this method is interpretable for arbitrary and unfixed shape. Circular fingerprints are a refinement of the Morgan algorithm [7] which helps to inference current stable structures in spite of atom reshaping.

#### fuseGNN: Accelerating Graph Convolutional Neural Network Training on GPGPU [3]:

This paper demonstrated a practical solution to improve two phases of GNN as Aggregation and Graph Processing. Because these two steps take much time and hardware to process, they propose to fuse GNN from PyTorch which is highly optimized APIs and CUDA kernel. Kernel-fusion accelerates 5.3X for both forward and backward propagation processing and make GNN more efficient on large datasets.

# Background

## Graph Convolutional Network

GCNN tries to gather information from the graph neighborhood and by the aid of Laplacian transform finds eigenvectors. Laplacian is proper for converting function of variable to function of complex variable, (Complex variable and Laplacian are mostly used for geometric combinations when they have mutated suddenly). This eigenvector or characteristic vector takes substantial data in order to pick up valuable data instead of redundant data.[13]

Then Fourier Transform helps us to depict a Graph Convolutional from the prior eigenvector. Fourier converts function of time or space as input from Laplacian into function of temporal or special frequency. Later popular Convolutional regularized functions helps to avoid overfitting and detect perfect features. Eventually all weights will be upgraded through backward propagation.

## Adam Optimizer

Adam is an optimizer that upgrades network weights. Its abbreviation comes from Adaptive Moment Estimation. Adam is applied because of making model efficient in both computation and implementation. Its hyper parameters need little tuning and accelerate implantation. Actually Adam is a substitute for Stochastic Gradient Descent (SGD) which is a traditional optimizer.

The fundamental difference between Adam and SGD is that Adam is good for sparse gradient and noisy problem and Adam change its learning rate during training time. Adam is a combination of Adaptive Gradient Algorithm (Adagrad) and Root Mean Square Propagation (RMSProp). Adam is suitable for none-stationary and large dataset which are changed arbitrary. [11]

## CUDA/GPU

Compute Unified Device Architecture (CUDA) is introduced in November 2006 by NVIDIA. CUDA is a platform for parallel computing which provides Application Programming Interface (API) for programmers. CUDA uses facilities on Graphic Process Unit (GPU).

Indeed, CUDA platform is a software layer that gives direct access to the set of virtual GPU instructions and parallel computing elements. This platform has been designed for C, C++, FORTRAN and Matlab languages. This direct accessing paves the ground for parallel programmers to use GPU resources.

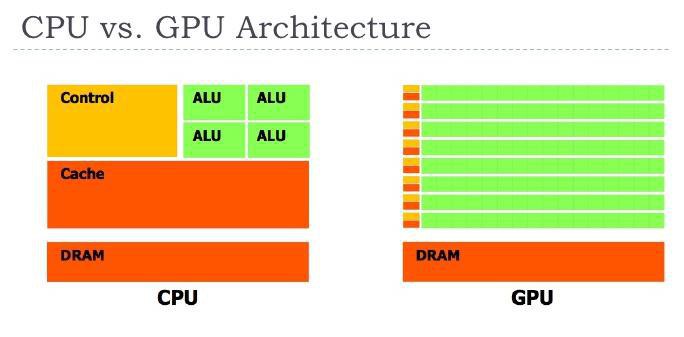
**CUDA has been designed for different purposes:**

1. Adding small distributions to the standard programming languages such as C which makes it possible to implement parallel algorithms without intermediate. Direct access or immediate implementation is to reduce hardware and software interface layer. Then, programmers can concentrate on the parallel algorithms instead of involving on the complex implementation.[8]

2. CUDA supports heterogeneous processing that programs can benefit from both CPU and GPU. Sequential and serial sections of programs can be executed by CPU and parallel parts by the aid of GPU. In fact CPU and GPU are distinct units which have their own separate memory. This kind of configuration allows simultaneous processing occurs on CPU or GPU without interferences.

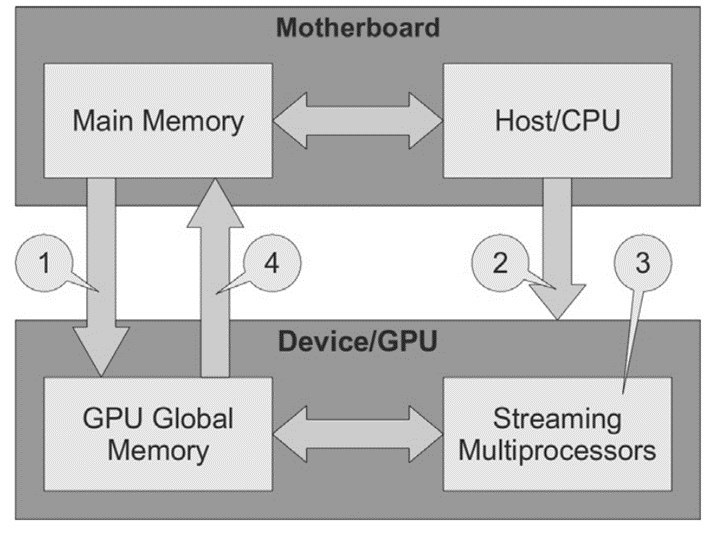
**CPU versus GPU Architecture:**

There are differences between CPU and GPU architecture. There are one or a few cores. Each core endure ponderous computation. But there are thousands of computation cores in GPU architecture that mostly execute parallelly parts of programs.



**Figure1: CPU vs. GPU Architecture. (a)CPU; (b)GPU. [8]**

There are hundreds of cores which perform simple tasks instead of CPU with four cores and one control unit, one cache and one dynamic ram. Actually every multi cores constitute one multi processing and each of them has dedicated memory including shared memory, local memory and register memory. Also, each multi processor has one controller and one Dynamic Ram which is used for I/O kernel operations.[3]

****

**Figure 2: CUDA Process. [10]**

**CUDA Process**

1. Data transition to GPU memory.

2. Transferring instructions for CPU to GPU.

3. Parallel processing in GPU unit.

4. Transfer result to the main memory.

### Access levels to the resources in GPU:

###### Thread level:

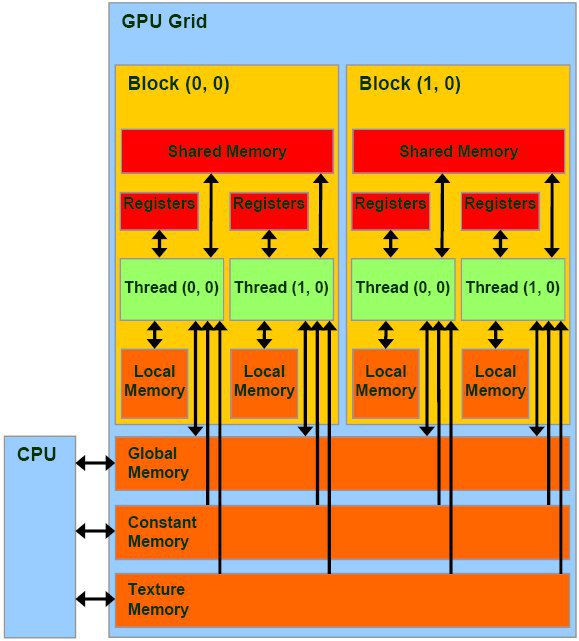
Every thread can access exclusively to the series of register memories which are as on-chip and local memory which are off-chip and un-cached. Register memories are on-chip, so delay is a few and bandwidth is high. Indeed, registers are the fastest memories which are 10 times faster than shared memory. Register lifetime is only on the thread.

###### Block level:

Every block is constituted from multiple threads. Each block exclusively has access to shared memory which are on-chip with small size and high speed.

###### Grid Level:

Every grid is organized from multiple blocks and has access to a global memory which is off-chip with big size and low speed. Global memory mostly will be used for I/O kernel due to its low speed.



**Figure 3: Heterogeneous CPU/GPU Architecture. [10]**

# Methodology

## Preprocessing

Laplacian is proper for converting function of variable to function of complex variable. We used it firstly to find eigenvectors or characteristic vector takes substantial data in order to pick up valuable data instead of redundant data. Laplacian are mostly used for geometrical and none stationary graph because dataset will be mutated suddenly.

The output of the Laplacian function will be the input for Fourier Transform. Fourier converts function of time or space as input from Laplacian into function of temporal or special frequency.

## NVIDIA GPU Cloud (NGC)

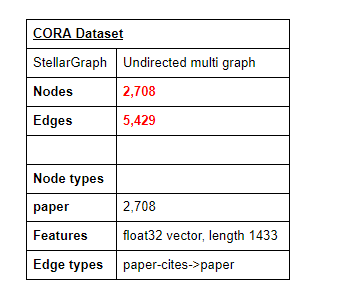
NGC has been applied for this research which is a GPU-accelerated cloud platform optimized. It takes full advantage of NVIDIA GPUs in single GPU or multi-GPU configurations. Google AI Hub and Colab is including GPU optimizations from NVIDIA CUDA-X. It is free and is available from <https://colab.research.google.com/>.

We just need to install some driver related to CUDA and after that we can use its spotless and widespread facilities freely. The only limitation is that each notebook has been valid for 12 hours per day.

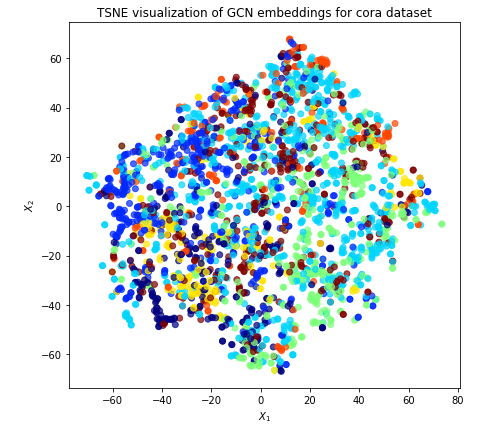
## Dataset

CORA includes of academic publications as the nodes or vertices and the citations between them as the links or edges. This dataset is very suitable for Graph Convolutional Network due to its none-stationary shape during time pass.

The Cora dataset contains of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words. [9]



**Figure 4: CORA Citation Dataset Spesifications. [9]**



**Figure 5: CORA Citation Dataset Graph Network. [9]**

## Parallel Computing

#### Kernel Function

Indeed, kernel is similar to C functions just with the difference to be applied as parallelly. All the threads in GPU work as parallelly, and simultaneously execute one instruction on the multiple data.

Therefore, the meaning of the parallel computing is that after invoking kernel with the size of dedicated threads will make a copy from the program and give to each thread one copy depends on program nature and size, will be applied by x, y, z components on each thread, and the specified instruction will be done by programmer. A Set of threads constitute one block and set of blocks constitute one grid. [8]

In order to define the function which should be executed on the GPU, and parallelly we must use \_\_global\_\_ on the top of the function definition. Kernels have no outputs and its return must be defined as void. Input arguments for kernels should be defined as pointers which point to the global memory.

In this research we defined all the loss function -forward and backward propagation- which are applied for generalizing the model as kernels and execute on the GPU device. These functions are drop out, sparse matmul, graphsum, relu, dense matmul and cross entropy.

For invoking kernel functions in the host function -host functions execute on the CPU memory- we use << >> the CUDA compiler will automatically define a variable as dim3 and define the number of block in the grid and number of threads in the block, as <<#blocks in grid, #threads in block>>.

## Propagation Function

Later popular Convolutional regularized functions helps to avoid overfitting and discover perfect features. Finally, the whole weights will be updated through backward propagation. These functions are drop out, sparse matmul, graphsum, relu, dense matmul and cross entropy which all are used for reducing overfitting. [12]

#### Drop out

Drop out is applied for dense fully connected layers such as Recurrent Neural Network (RNN), Long Short Term Memory (LSTM), Convolutional Neural Network (CNN). Dropout remove randomly the connection between nodes, and it causes increasing weight, so there is necessity to recalling weight after finishing dropout. It increases running time but because of regularization create better model. [14]

#### Sparse Matmul

Sparse Matrix Multiplication is another solution for regularization matrices. It includes mostly by zero value and is multiplied to a graph matrix which concludes to have a generalized model. In python there is a function as sparse.csr\_matrix inside scipy library which helps us to create matrix but here in this research due to C++ language we created it.

#### RelU

Rectified Linear Unit (ReLU) is activation function for most of neural networks such as RNN, LSTM, CNN, GNN. RelU has a better performance rather than previous activation functions due to its linear functionality to input data. In this research we defined a function which simple keep value if they are greater than zero "bool keep = in->data[i] > 0", "if (training) mask[i] = keep;" and pass away if they are zero: "if (!keep) in->data[i] = 0;"

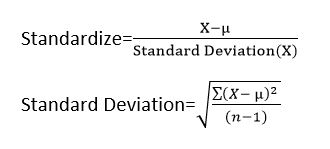
#### Cross Entropy

Cross Entropy computes probability distributions between graph and random variable. Cross entropy will get zero if the model is perfect accurate which shows us the model is overfitting. Its formula is the sum of logarithm output. [15]

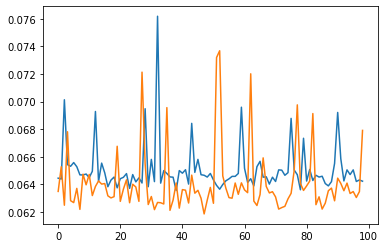
## Evaluation

In this section we demonstrate how much adam optimizer helped us to enhance Graph Convolutional Network. We can observe output concretely with high resolution graph which have been drawn by matplotlib in python. Judging about optimizer can be easy after gathering results for at least 100 epochs for both C++ and CUDA on GPU with or without optimizer. Each graph has a legend in the upper right corner and described x-axis as epoch number and y-axis says about time as seconds.

Because we had a dramatically changes on CUDA/GPU and the scattering will be in disorder (Figure [7, 8]), therefore we need to normalize or scaling the value in order to have equal range and/or variance. To standardize the value in each experiment we use the standard deviation and below formula.

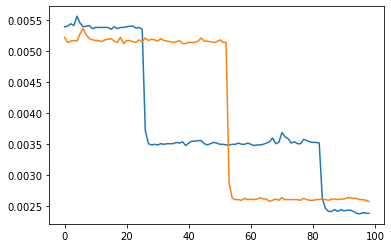


**Figure 6: Standardize CPU/GPU C++/CUDA outputs.**

****

**Figure 7: C++ process on GPU/ With and Without Adam**

**(0.062 < running time < 0.076)**

****

**Figure 8: CUDA process on GPU/ With and Without Adam**

**(0.0025 < running time < 0.0055)**

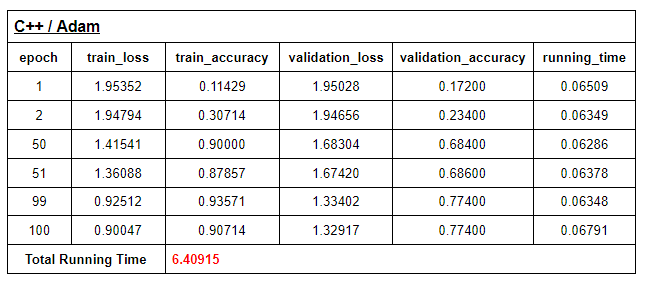
The main goal in this research is to discover that whether parallel computing by CUDA platform and optimizer boost model performance in both running-time and accuracy. In order to figure it up if we are in the right direction or not is to calculate related parameters during each epoch. These parameters called running-time, train-loss, validation-loss, train-accuracy, validation-accuracy. In order to have a perfect model which can predict almost any new values, we need to have a general model. Massive data causes to overfitting means that model matches completely train set to test data. While this model is unable to anticipate an answer for unseen situation and data.

Cross validation is a solution to prevent overfitting which is a model evaluation method. We divide the train set to K-fold and in each iteration, kth part belongs to the test and the rest of k-1 is the train set, therefore the chance of matching will be decreased. There are substitute functions for cross-validation in convolutional neural network in order to avoid overfitting such as drop out.

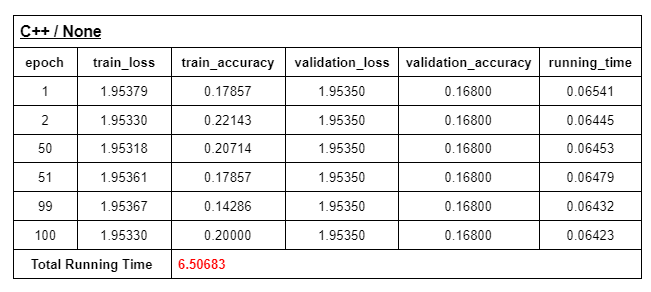
The kth part evaluates the model's performance and has been shows as validation-accuracy, it says about accuracy about new data which will be injected in the future. When train-accuracy is high and validation-accuracy is low illustrates that the model is fitting perfectly while validation parts can not reach to fit as well as train, this paradox indicates that the model is starting to fit on nose data and finally gets overfitting.

The differences between what we want and what the model really can predict can be measured by cost function or loss function. Our aim is to keep downward validation-loss for preventing overfitting. Equality between validation-loss and train-loss proves us to reconsider our model to avoid high overfitting.

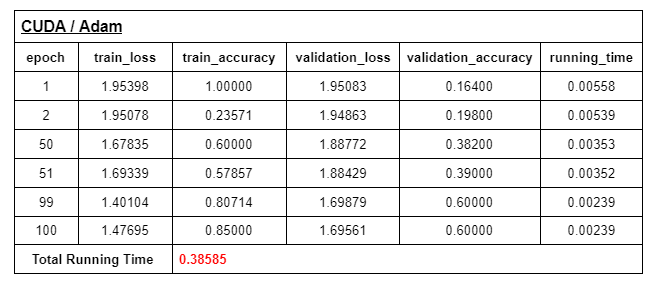
## Experiments



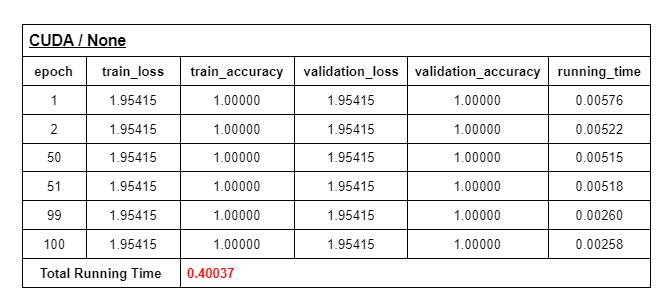
**Table 1: C++/ Adam**



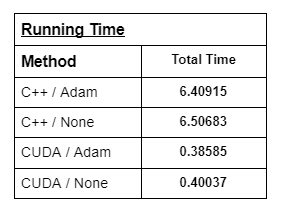
**Table 2: C++/ None**



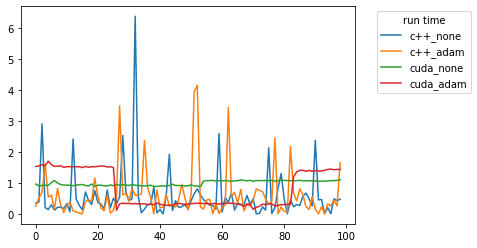
**Table 3: CUDA/ Adam**



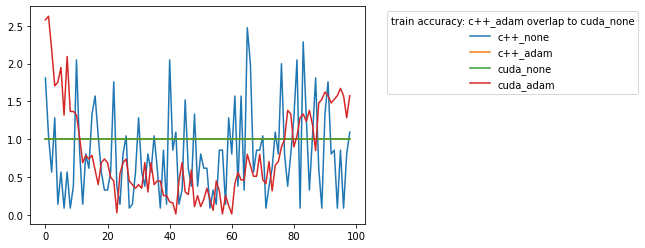
**Table 4: CUDA/ None**



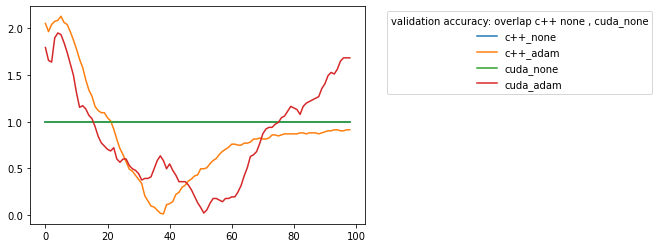
**Table 5: Total Running Time**



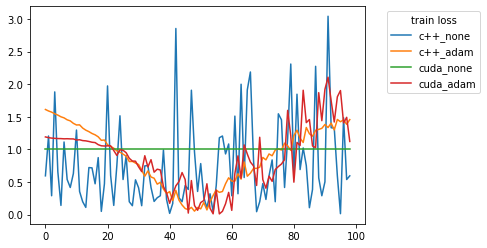
**Figure 9: Running Time**



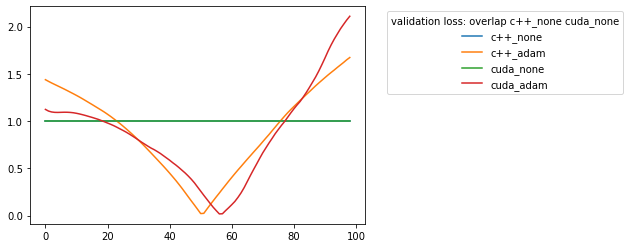
**Figure 10: Train Accuracy**



**Figure 11: Validation Accuracy**



**Figure 12: Train Loss**



**Figure 13: Validation Loss**

# Conclusion and Future Research

We extended the state-of-the-art We extended the state-of-the-art Graph Convolutional Neural Network on C++ language and CUDA platform. The result shows us how much program will be speed when using CUDA/GPU because of parallel programming and heterogeneous functions. We obtained very promising results in Tables[1, 2, 3, 4, 5]. Table[5] illustrate how much Adam accelerates C++ code and time will be diminished from 6.50683 to 6.40915 and Adam enhanced C++ code performance about 1.5 percent. This story will happen for CUDA and Adam could improve CUDA code performance about 5 percent.

Comparison between CUDA and C++ illustrated to us that how much using heterogeneous processors and parallel computing can be helpful. The observations showed us running time decreases from 6.5 to 0.3, it is almost 95.5 percent improvement which in running time.

By a glance to train and validation accuracy and their loss values, we figure it up that Adam causes a bit error that it could be because of none stationary graph and learning rate. There is less error in SGD, but Adam suggests better running time. In the future, we can hybridize these two optimizer and thinking about a solution to decrease error.

# References

[1] Gómez-Luna, J., El Hajj, I., Chang, L. W., García-Floreszx, V., de Gonzalo, S. G., Jablin, T. B., ... & Hwu, W. M. (2017, April). Chai: collaborative heterogeneous applications for integrated-architectures. In Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on(pp. 43-54). IEEE.

[2] Gupta, P., Gasse, M., Khalil, E. B., Kumar, M. P., Lodi, A., Bengio, Y. (2020, June). ‘Hybrid Models for Learning to Branch’. *34th Conference on Neural Information Processing Systems.* Vancouver, Canada, 2020. Advances in Neural Information Processing Systems 33.

[3] Z. Chen et al., "fuseGNN: Accelerating Graph Convolutional Neural Network Training on GPGPU," *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), San Diego*, CA, USA, 2020, pp. 1-9.

[4] J. Nickolls, "Scalable parallel programming with CUDA introduction," 2008 IEEE Hot Chips 20 Symposium (HCS), Stanford, CA, USA, 2008, pp. 1-9, doi: 10.1109/HOTCHIPS.2008.7476518.

[5] D. Duvenaud, D. Maclaurin, J. A. Iparraguirre, R. G. Bombarelli, T. Hirzel, A. A. Guzik, R. P. Adams, “Convolutional Networks on Graphs for Learning Molecular Fingerprints”, Neural Information Processing Systems (NIPS), 2015.

[6] M.Niepert, M.Ahmed,K.Kutzkov, “Learning Convolutional Neural Networks”, International Conference on Machine Learning (ICML), 2016.

[7] Robert C. Glem, Andreas Bender, Catrin H. Arnby, Lars Carlsson, Scott Boyer, and James Smith. Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to ADME. *IDrugs: the investigational drugs journal*, 9(3):199–204, 2006.

[8] D. R. V. L. B. Thambawita, R. Ragel and D. Elkaduwe, "To use or not to use: Graphics processing units (GPUs) for pattern matching algorithms," *7th International Conference on Information and Automation for Sustainability*, Colombo, Sri Lanka, 2014, pp. 1-4, doi: 10.1109/ICIAFS.2014.7069585.

[9] Graph Convolutional Network (GCN) on the CORA citation dataset. *stellargraph.readthedocs.io*. 2018-2020, Data61, CSIRO Revision 625a99b7.

<https://stellargraph.readthedocs.io/en/v1.0.0rc1/demos/node-classification/gcn/gcn-cora-node-classification-example.html>.

[10] T. Okuyama, F. Ino and K. Hagihara, "A Task Parallel Algorithm for Computing the Costs of All-Pairs Shortest Paths on the CUDA-Compatible GPU," *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, Sydney, NSW, Australia, 2008, pp. 284-291, doi: 10.1109/ISPA.2008.40.

[11] M. R. Izadi, Y. Fang, R. Stevenson and L. Lin, "Optimization of Graph Neural Networks with Natural Gradient Descent," *2020 IEEE International Conference on Big Data (Big Data)*, Atlanta, GA, USA, 2020, pp. 171-179, doi: 10.1109/BigData50022.2020.9378063.

[12]Kipf, T., Welling, M. Graph Convolutional Networks <https://github.com/tkipf/gcn>.

[13] Kipf, T., Welling, M. “Semi-Supervised Classification with Graph Convolutional Networks. *5th International Conference on Learning Representations (ICLR-17).*

[14] Srivastava., N., Hinton., G., Krizhevsky., A., Sutskever., I., Salakhutdinov., B. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014) 1929-1958.

[15] Json Brownlee, A Gentle Introduction to Cross-Entropy for Machine Learning.October 2019. *https://machinelearningmastery.com/cross-entropy-for-machine-learning/*