LITERATURE REVIEW: — Parallelizing the Interpolation between Latent Space of Autoencoder Networks to Introduce Novelty in 3D Object Reconstruction

Tansin Jahan
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
tansinjahan@cmail.carleton.ca

October 5,2018

1 Introduction

In terms of hardware, running every single instruction relies somehow on the computation of either CPU or GPU. Based on the capability of both CPU and GPU, runtime of the program is defined and when this runtime becomes an overhead, it is required to introduce some improvements so that the computation complexity can be minimized. This is where **parallelization** weigh in as it is capable of utilizing GPU for heavy computation of data.

My project involves parallelizing the vector addition produced by a simple **Autoencoder Neural Network** which is popular to reconstruct same 3D object given as an input. Autoencoder, being a symmetrical deep network involves several convolutional layer and parameters for computation. The implemented CNN for this project feeds on 3D vol-

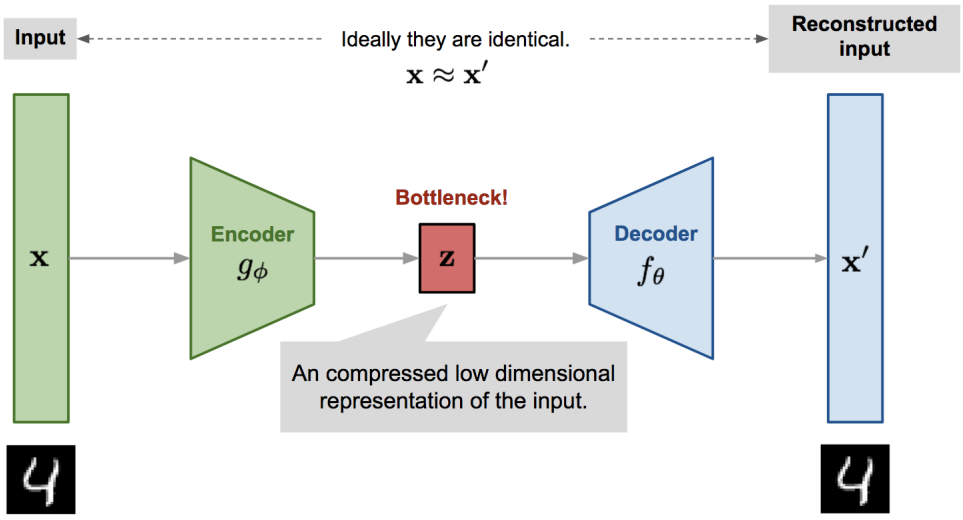

Figure 1: A Simple Autoencoder Architecture

umes as input involving higher dimension(i.e. 32x32x32) and then reduces that into lower dimension [7] which is called as latent space(also Z). In latent space, represented as Z vector, the input object has minimum dimension with maximum features. From the lower dimension, the Z vector passes to deconvolution and therefore produces the same 3D volumes. But we can combine Z vector of two input volumes and interpolate new points as Z vector so that decoder can produce new 3D objects based on this interpolation. This interpolation of Z vector (addition of two Z vectors) can be an overhead for the performance of the Autoencoder. Let us consider we have 50 inputs. Then for each input, we have to compare it with another 49 inputs and calculate interpolation each time. So, for 50 inputs the vector addition will be 50 x 50 which is in total 2500 times addition of vectors.

The above calculation can take much time compared to the convolution and therefore we can parallelize this computation in GPU by using Multithreading option. In conclusion, for this project several thread will be introduced to compute vector addition in CUDA so that the performance of the whole network can be improved.

2 Literature Review

With the recent development of Convolutional Neural Network, it has been used to solve several Computer Vision problem. For example, object detection or reconstruction from input image, semantic information extraction from a scenario, object segmentation- these are all recent computer vision application where CNN has been used to produce better result. Likewise, 3D reconstruction of an object is one of the Computer Vision problem and recently multiple approaches (ex: 3D-GAN, 3D-shapenets) has been proposed as a solution to the problem [5]. Previous work shows that given a depth image as input, the volumetric representation can be produced [6]. Following these works, an approach to reconstruct 3D voxelized object from different viewpoint of one or more images of that object (i.e. single-view or multi-view) is proposed where recurrent neural network has been used [1]. In total 50,000 model is used to train and test the proposed network. Training the network with this large amount of data is really time consuming and therefore introducing parallelism between the layers of the model can help to improve the performance of the network.

2.1 GPU implementation

GPUs are capable of efficiently running parallel programs and outperforms CPUs in terms of raw computing power. In the paper, 'Imagenet classification with deep convolutional neural networks' - two GTX 580 3GB GPU has been used to train the network which took five to six days to complete the whole training [2]. In this parallelization scheme, they kept half of the kernels (or neurons) on each GPU, where the GPUs communicate only in certain layers.

GPU's parallel processor architecture has made it an essential choice for several applications where parallelism is needed. Such most common areas where GPU computing can be used are - Bioinformatics, Data Science, Analytics, and Databases, Defense and Intelligence, Machine Learning, Imaging and Computer Visions etc. [4]. Using GPU's multithreaded processors, several threads can be introduced to perform matrix operations which is highly efficient for both graphics and general-purpose parallel computing applications [3]. To execute programs in GPU written by high-level languages such as C, C++ or Python - CUDA provides a high level interface by dividing CPU as host memory and GPU as device memory.

References

[1] Christopher Bongsoo Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *CoRR*, abs/1604.00449, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. In *ACM SIGGRAPH 2008 classes*, page 16. ACM, 2008.

[4] Bhavana Samel, Shubhrata Mahajan, and AM Ingole. Gpu computing and its applications. *International Research Journal of Engineering and Technology (IRJET)*, 3(04), 2016.

[5] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, and Joshua B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *CoRR*, abs/1610.07584, 2016.

[6] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[7] Zhuotun Zhu, Xinggang Wang, Song Bai, Cong Yao, and Xiang Bai. Deep learning representation using autoencoder for 3d shape retrieval. *Neurocomputing*, 204:41–50, 2016.