Exploration of System Configuration in Effective Training of CNNs on GPGPUs

Li Zhang[1], Xianwei Cheng[1], Hui Zhao[1], Saraju P. Mohanty[1], Juan Fang[2]
[1]Computer Science and Engineering Department, University of North Texas
{lizhang2,xianweicheng}@my.unt.edu, {hui.zhao,saraju.mohanty}@unt.edu
[2]Faculty of Information Technology, Beijing University of Technology
fangjuan@bjut.edu.cn

***Abstract*—Convolutional Neural Networks (CNNs) have shown a great potential in different application domains including object detection, image classification, natural language processing, and speech recognition. Since the depth of the neural network architectures keep growing and the requirement of the large-scale dataset, to design a high-performance computing hardware for training CNNs is very necessary. In this paper, we measure the performance of different configuration on GPU platform and learning the patterns through training two CNNs architectures, LeNet and MiniNet, both perform the image classification. Observe the results of measurements, we indicate the correlation between L1D cache and the performance of GPUs during the training process. Also, we demonstrate that L2D cache slightly influences the performance. The network traffic intensity with both CNN models shows that each layer has distinct patterns of traffic intensity.**

I. INTRODUCTION

Deep Learning techniques became more and more popular, because it has shown a great deal of success in serval domains including object detection, image classification, nature language processing, and speech recognition [1]. However, the fundamental ideas have stated about three-decade ago [2]. The current success of deep learning benefited from two main reasons: 1) the development of the Internet, which produces massive number of data; 2) the computation capability of hardware grows rapidly, which can deal large-scale training data efficiently.

Deep learning is one class of machine learning algorithms, which can train a non-linear function model by multiple layers of neurons. In this paper, we only consider a special class of deep learning architectures, called Convolutional Neural Networks (CNNs). CNNs is good at object detection, image classification, natural language processing, and speech recognition.

CNN architectures are composed of several layers and each layer takes three-dimensional data as input then implements a three-dimensional filter or weight to produce a three-dimensional result which will be the input for the next layer. The first layer corresponds to input data and the output of the last layer corresponds to the predicted output. These filter or weight for each layer can be updated by backpropagation through the learning process. Through the forward pass with current weight, the predicted result can be produced, then compare to the label and gain the gradient of prediction error. Through the backpropagation, the gradient of prediction error passed backwards to the network. By using these gradients, the weight of each layer can be updated [2].

Training speed of CNNs highly depends on the computation capability of hardware and the size of the dataset. Since the most computations of CNN training are involve matrix dot products and vector dot product at each layer. Good news is, these computations exhibit data parallelism. A GPU-based manycore platform is more suitable for these tasks, and it can significantly accelerate the training speed [3]. So, it's the most preferred choice to implement different types of deep neural network architectures. TensorFlow and Caffe are two popular frameworks for deep learning and they both implemented in GPU-based systems [4][5].

The remainder of the paper is organized as follows. In Section 2, we provide an overview of the CNNs and the details of a training process. Also, we present the dataset we use and the architectures we choose. In Section 3, we discuss the results of the experiment to demonstrate the correlation between each factor and performance. Finally, Section 4 concludes the paper by summarizing the results and pointing out the directions of the future work.

II. CONVOLUTIONAL NEURAL NETWORKS

In this section, the fundamental ideas of CNNs and the mathematic theory will be stated

A. Overview of CNNs

A typical CNN architecture is composed of serval different layers which including convolutional layers, pooling layers, and fully connected layers. Convolutional layer: The objective of a Convolutional layer is to extract features of the input volume. Which means transform a low-level representation into a high-level representation. Also, we only connect part of the image to the next layer because if all the pixels of the input are connected, it will be too expensive. During the forward pass, apply dot products between a receptive field and a filter and produces a feature map. Then we slide the filter overall receptive field with the same filter and generate a set of feature

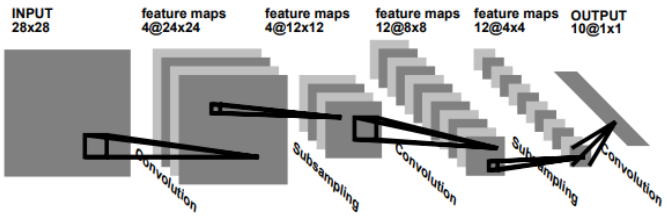

Figure 1. Overview of the CNN architecture for digit classification [6].

Table 1. Layer configurations for LeNet

Layer	Input	Filter	Output
Conv1	28*28*1	5*5*6	24*24*6
Pool1	24*24*6	2*2*1	12*12*6
Conv2	12*12*6	5*5*6*16	8*8*16
Pool2	8*8*16	2*2*1	4*4*16
Conv3	4*4*16	4*4*16*120	120
Fully1	120	120*86	86
Fully2	86	86*10	10

map. After that, an element-wise non-linear activation function will be applied to the set of feature map and because of input of the next layer.

Pooling layer: Normally, there's two types of pooling layer base on the operation it uses. Max operation generates max value from the receptive field and average operation generates average value from the receptive field. Pooling layer performs a function to reduce the spatial dimensions of the input, and the the computational complexity of the model. Also, it can avoid the overfitting.

Fully connected layer: Fully connected layers always at the end of the network, since the input already though the convolutional layers and pooling layers, the volume of neuron has significantly reduced. So, this layer can connect every neuron in one layer to every neuron in another layer. The last fully-connected layer classifying the generated features of the input image into various classes based on the training dataset.

B. Dataset and Architectures

In the work, we consider one of the most widely used image classification dataset, namely MNIST [7]. The MNIST database of handwritten digits, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

We used two architecture, namely LeNet and MiniNet, and trained with MINST dataset.

LetNet: exclude the input layer and output layer, LetNet has seven layers, which are three convolutional layers and two max pooling layers and two fully connected layers. The configuration for each layer shows in Table 1 [8].

MiniNet: exclude the input layer and output layer, the architecture only has three layers, so we call it MiniNet. These three layers are convolutional layers, max pooling layers and fully connected layers and the configurations of each layer can be found in Table 2. Both neural networks are trained by MNIST dataset. The LeNet is more complex than the MiniNet. LeNet has three convolutional layers and has total 238 filters, MiniNet only has one convolutional layer and has 17 filters.

Table 2. Layer configurations for MiniNet

Layer	Input	Filter	Output
Conv1	28*28*1	5*5*6	24*24*6
Pool1	24*24*6	4*4*1	6*6*6
Fully1	6*6*6	6*6*6*10	10

C. Training Procedures

Training a CNN in another word, is training the weights of each layer of CNN. Like we mentioned above, to update the weights, the backpropagation algorithm is used. Before that, we need to get the prediction error from pass forward with current weights.

Perform a pass forward on the convolutional layer as follows:

$$a^{(l)} = \sigma\left(z^{(l)}\right) = \sigma\left(a^{(l-1)} * W^{(l)} + b^{(l)}\right) \quad (1)$$

Which $a^{(l)}$ is an output of layer l, σ is activation function, $W^{(l)}, b^{(l)}$ is weight and bias of layer l. $*$ indict the convolution operator.

Perform a backpropagation on the convolutional layer as follows:

$$\delta^{(l)} = \left(\left(W^{(l)}\right)^T \delta^{(l+1)}\right) \cdot \sigma'\left(z^{(l)}\right) \quad (2)$$

$$\nabla W^{(l)} = \delta^{(l+1)}\left(a^{(l)}\right)^T \quad (3)$$

$$\nabla b^{(l)} = \delta^{(l+1)} \quad (4)$$

Which $\delta^{(l)}$ is error for an l-th layer. $\nabla W^{(l)}$ and $\nabla b^{(l)}$ are gradients for the l-th layer.

Base on the equation (1), (2), (3), (4), obviously, the pass forward and backpropagation require lots of matrix computations and a lot of data parallelism exist.

For both pass forward and backpropagation, convolutional layers contain the most expensive operations. Each convolution operation requires multiple dot products of the receptive field and filter values.

III. EXPERIMENTAL RESULTS AND ANALYSIS

To do this experiment, we used GPGPU-sim, a well-known detailed, cycle-level simulator modeling contemporary graphics processing units (GPUs) running GPU computing workloads written in CUDA or OpenCL [9] [10][11]. We customize the configuration of GPGPU to exam the correlations between different factors. Table.3 shows the details of the configurations.

Form the Fig.2 to Fig.9. left part of the figures presents the forwarding pass, and right part of the figures presents the backpropagation. And the bar of "c" presents the IPC number

Table 3.System configuration

Configuration	L1D	L2D	Network
Baseline	32	64	Butterfly
Perfect	32	64	Perfect
L1D_32_L2D_32	32	32	Butterfly
L1D_64_L2D_32	64	32	Butterfly
L1D_64_L2D_64	64	64	Butterfly

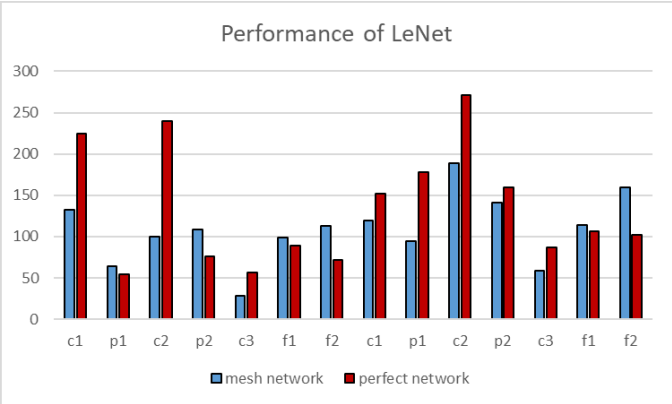

Figure 2. IPC for training the LeNet with Mesh Network and Perfect Network.

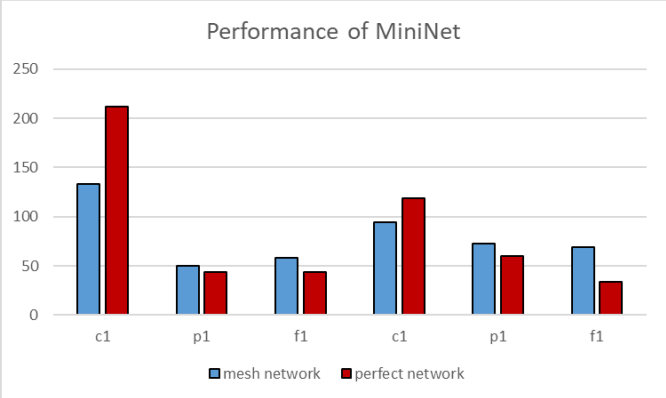

Figure 3. IPC for training the MiniNet with Mesh Network and Perfect Network.

of convolutional layer, "p" means pooling layer and "f" means fully connected layer.

A. Mesh Network vs Perfect Network

According to the Fig.2 and Fig.3, we see the performance of the Perfect Network is better than Mesh Network during the convolutional layers, however, the Mesh Network perform better than Perfect Network when computing the fully connected layers. Also, we can see, when the computation size increase, the performance of Perfect Network gives significantly raise. Since the Perfect Network doesn't have any delay during on-chip data transmission, so the result indicates the convolutional layers request lots of data transmission, and fully connected layer request less data transmission.

B. L1 Cache Configuration

According to Fig.4 we can see, when the L1D cache increase, the performance in the backpropagate stage is better. Especially, the performance of max pooling layer 1 (denote as

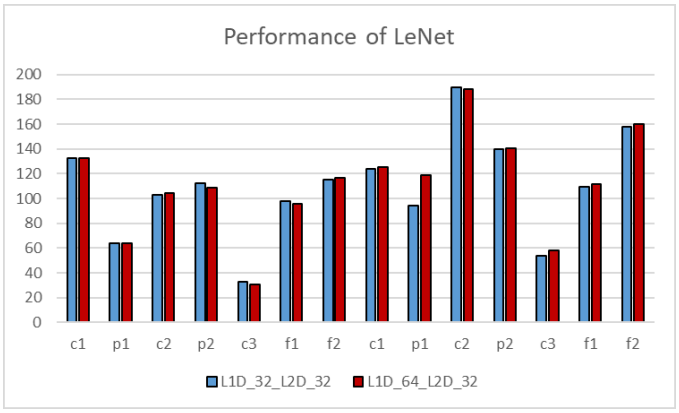

Figure 4. IPC for training the LeNet with different L1D cache Configuration.

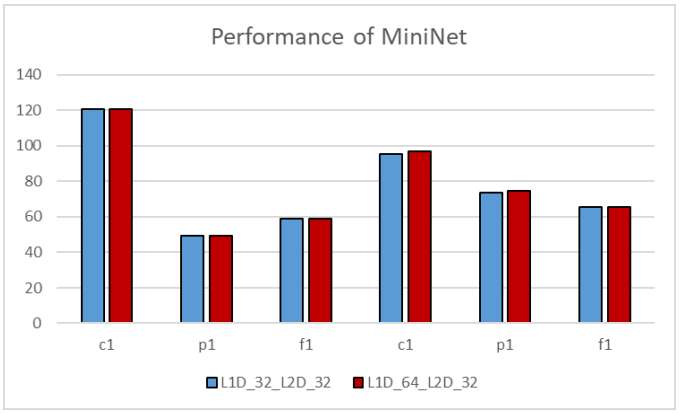

Figure 5. IPC for training the MiniNet with different L1D cache Configuration.

p1) raised significantly. Also, Fig.5 shows the same result as the Fig. 4 except the change rate is smaller than Fig.5. So, we think the L1D cache influence the performance of CNN. Which increase the L1D cache will raise the performance of both CNN architectures, especially, when the size of CNN architecture is large.

C. L2 Cache Configuration

To test that how L2 cache impacts the performance of both CNN architectures, we set up the different L2D cache. Fig.6 shows the performance of convolutional layer 1 (denote as c1) raised and convolutional layer 2 (denote as c2), max pooling layer 1 and 2 (denote as p1, p2) dropped， but the changes are very slight. In Fig.7, the performance of each layer in MiniNet is exactly the same. According to these experimental data, we can conclude since we double the L2D cache for both small and large CNN architectures, the performances almost remained the same, there is no correlation between IPC and L2D cache.

D. Analysis of Inject Rate

We compare the network traffic intensity (measured in normalized flits injection rate) with different CNN layers in Fig.8 and Fig.9. Both in forward pass and backpropagation, the figures show, that convolutional layers (denoted as c) have

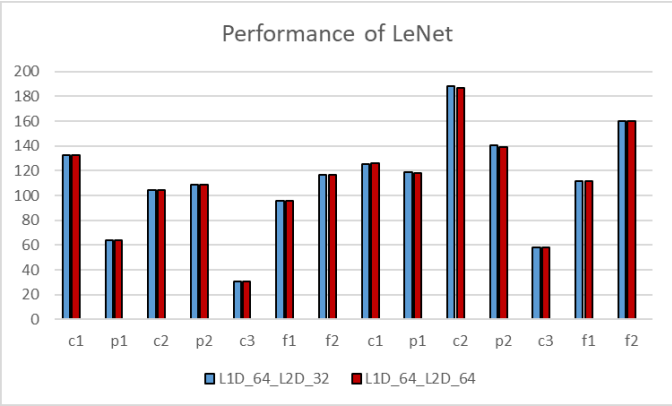

Figure 6. IPC for training the LeNet with different L2D cache configuration.

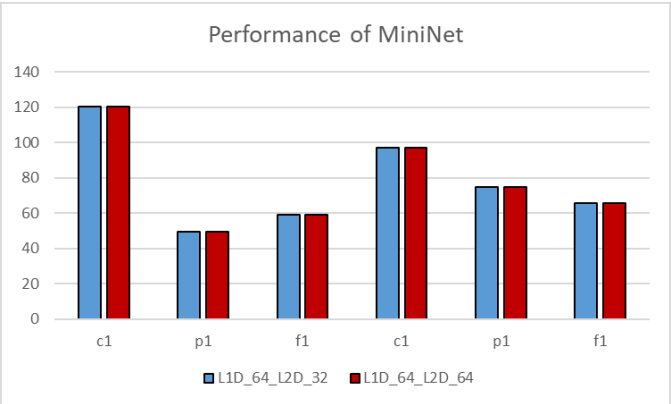

Figure 7. IPC for training the MiniNet with different L2D cache configuration.

highest injection rates. In Fig.8, the fully connected layer 1 (denoted as f1) has higher injection than the max pooling layer 2 (denoted as p2) during both forward pass and backpropagation. For the average, fully connected layers have the lowest injection rate, max pooling layers have higher injection rates than fully connected layers but lower than convolutional layers.

IV. CONCLUSION AND FUTURE WORK

The results of the experiment demonstrate the correlation between some factors and performance, they also indicate some factor which will not impact the performance of the GPU during the training process.

The results of the comparison between the perfect network and mesh network, concludes on-chip latency highly affects the performance since the data transmission is very active during the training process. We also indicate that L1D cache can impact the performance, since the IPC increase along with the L1D cache increase. L2D cache slightly influences the performance, especially the volume of computation is low, increase the L2D cache shows the same results as the regular L2D cache configuration. The network traffic intensity with both CNN models demonstrate that each layer of any CNN involves distinct computation patterns, the volume of the traffic varies from one layer to another. These unique patterns show the convolutional layer has the highest traffic intensity, and the fully connected layer has lowest traffic intensity.

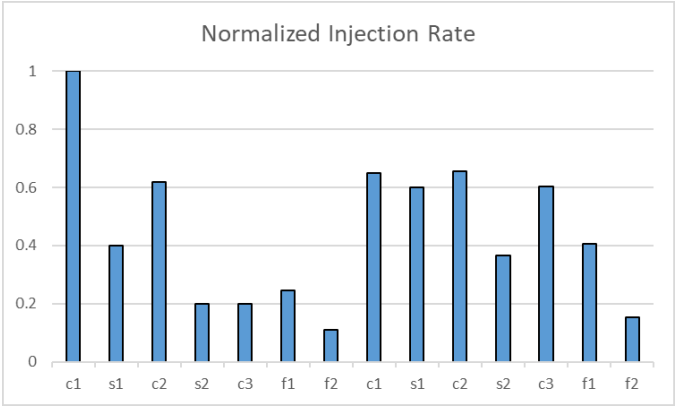

Figure 8. Normalized injection rate of LeNet.

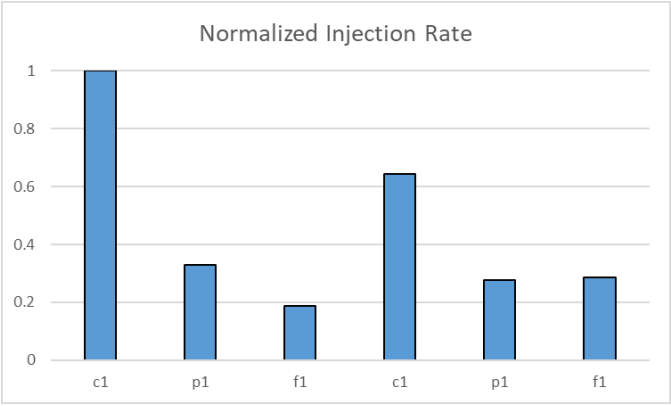

Figure 9. Normalized injection rage of MiniNet.

V. REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton. “Deep learning”. Nature 521: 436–444. 2015. J. Clerk Maxwell, *A Treatise on Electricity and Magnetism,* 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp. 68-73.

[2] M.Abadi, et al, “TensorFlow: Large-scale machine learning on heterogeneous systems”. Software available from tensorflow.org, 2015.

[3] Y. A. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Müller (Muller), E. Säckinger (Sackinger), P. Y. Simard, V. N. Vapnik, "Learning algorithms for classification: A comparison on handwritten digit recognition", Neural Networks, pp. 261-276, 1995.

[4] Y. LeCun, C. Cortes, “The MNIST database of handwritten digits”, http://yann.lecun.com/exdb/mnist/

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. “Gradient-based learning applied to document recognition”. Proc. of the IEEE, november 1998.

[6] Ali Bakhoda, George Yuan, Wilson W. L. Fung, Henry Wong, Tor M. Aamodt,“Analyzing CUDA Workloads Using a Detailed GPU Simulator” , in IEEE InternationalSymposium on Performance Analysis of Systems and Software (ISPASS), Boston, MA, April 19-21, 2009.

[7] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, Vijay Janapa Reddi,“GPUWattch: Enabling Energy Optimizations in GPGPUs”, In proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA 2013), Tel-Aviv, Israel, June 23-27, 2013.

[8] Aaron Ariel, Wilson W. L. Fung, Andrew Turner, Tor M. Aamodt, “Visualizing Complex Dynamics in Many-Core Accelerator Architectures”, In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 164-174, White Plains, NY, March 28-30, 2010.