Hardware Acceleration of Deep Learning

PI: Tao Xie
Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana, Illinois, USA 61801
Email: taoxie@illinois.edu
co-PI: Yuan Xie
Electrical and Computer Engineering Department, University of California, Santa Barbara
Santa Barbara, California, USA 93106
Email: yuanxie@ece.ucsb.edu
Maohua Zhu
Electrical and Computer Engineering Department, University of California, Santa Barbara
Santa Barbara, California, USA 93106
Email: maohuazhu@ece.ucsb.edu

***Abstract*—Our project aims at evaluating the Blue Waters platform for hardware acceleration of deep learning for big data image analytics and machine translation. To achieve near real-time learning, efforts must be paid for both hardware scaling out (increasing the number of compute nodes in a cluster) and scaling up (improving the throughput of a single node by inserting hardware accelerators). In this work, we evaluated the performance of scaling up using the GPU enabled node (XK7) for training convolutional neural networks. The key observation we got is that different types of neural networks (e.g. convolutional neural networks, long short-term memory) have different computation per byte so that current resource allocation approaches cannot achieve optimal efficiency for a multi-node system running multiple type of neural networks. Base on this observation, we will further model the correlation between the system performance and resource allocation and workload scheduling policy in the next generation work on the Blue Waters.**

I. Introduction

Deep learning has been widely used in applications such as image classification, speech processing and object recognition. The huge amount of training data required by the deep neural networks asks for more computing power to keep pace of the advance of state of the art accuracy of these tasks. Mainstream deep learning facilities are CPU-based clusters, which usually consist of thousands of compute nodes. As the major computation of deep learning is convolution and matrix multiplication, which is suitable for Graphic Processing Units (GPUs) to process, modern deep learning facilities are often equipped with GPUs as hardware accelerators. However, straightforward implementation of deep neural networks on such GPU enabled compute nodes will lead to under-utilization of compute resources, especially for multi-node systems such as the Blue Waters. Therefore, there is a strong motivation to evaluate and characterize the deep learning workload on the GPU-enabled nodes.

In this work, we evaluated the performance of popular types of deep neural networks on a GPU enabled supercomputer (XK7 nodes on the Blue Waters). From the evaluation results, we observed good scalability of neural networks on the Blue Waters supercomputer. Meanwhile, among the three types of neural network layers we evaluated, that is, convolutional layers, fully connected layers, and long short-term memory (LSTM) [1] layers, the convolutional layers have the best scalability. The difference among the three types of network layers in terms of scalability comes from the variation of computation per byte in each layer. Given the same number of weights, the convolution layers have one order magnitude larger number of multiply-accumulation (MAC) operations since the computation complexity of convolution is higher than matrix multiplications. Furthermore, the convolutional layers employ the weight sharing technique, which dramatically increases the computation per byte of the network.

Based on these observations, we continue to explore the design space of mapping different kinds of neural networks onto the GPU-enabled supercomputer. In real-world data centers, there are numerous neural network-based applications running concurrently. Since the optimal number of nodes allocated for each type of neural networks varies, we should design a scheduling method to achieve the best efficiency. In next generation of work, we will conduct more application characterization on the multiple-type neural network workload on the Blue Waters.

II. Workload Description

A. Convolutional Neural Networks

In the last several years, convolutional neural networks (CNNs) have achieved the state-of-the-art accuracy in various tasks such as image classification and object recognition, thanks to its high representational power. A convolutional neural network is a kind of deep neural networks which has one or more convolutional layers. Neurons in convolutional layers share connection weights so that the parameter space is much smaller than other kinds of deep neural networks. Therefore, it can be trained by back propagation method directly without any pre-training. However, as problems become more complex, CNN solutions require larger networks and more training data. For example, the ILSVRC-2012 winner AlexNet [2] has 60 million parameters and 650,000 neurons, consisting of five convolutional layers. Two years later, the ILSVRC-2014 winner VGG-16 [3] improved the accuracy by another 10% with 138 million parameters. And this trend of

the growing size of networks will continue as the amount of data keeps increasing exponentially. It usually takes weeks to a month to train such a big model even with the help of a thousand-node cluster. Furthermore, for a neural network that is customized for a specific problem, some training hyper-parameters like learning rate and momentum have to be swept to ensure the network to converge on the global minimum, which makes the training process even longer. To accelerate the training of neural networks, GPUs are widely used as accelerators for both single-node and cluster-based deep learning systems.

B. Long Short-Term Memory

Long Short-Term Memory (LSTM) is one popular type of recurrent neural networks (RNNs) used in speech recognition and machine translation. LSTM has a more complex neuron cell structure than CNNs.

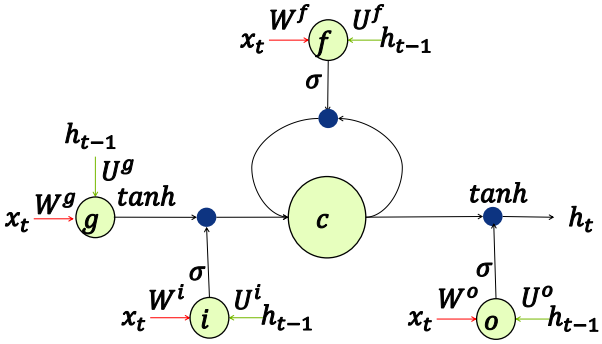

Fig. 1: Basic LSTM cell

The LSTM cells used in this paper are all basic LSTM cells. For each cell, the forward propagation flow is as below:

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i)$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f)$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o)$$

$$g_t = tanh(W^g x_t + U^g h_{t-1} + b^g)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$

$$h_t = o_t \circ tanh(c_t)$$

As shown in Figure 1, i_t, f_t, and o_t stand for input gate, forget gate, and output gate, respectively. These sigmoid-based gates (σ stands for sigmoid) are used to prevent irrelevant input from affecting the memory cell (c_t). The new cell state (g_t) is a preliminary summary of the current input from the previous layer and the previous status of current layer. The final hidden status h_t is the output of the LSTM cell if it is seen as a black box.

C. Distributed DNN Workload

Distributed neural networks allocate one compute node for each replica, which is a part of the training model. The training process of each layer in each replica consists of three phases:

1) Receiving output data from the previous layer of other replicas and feeding them into the device memory;
2) Executing the device kernel to compute the output data of current layer;
3) Sending the output data of the current layer to other replicas.

The performance bottleneck of this straightforward implementation is that Step 1 and 3 introduce very long latency, which is determined by the longest unpredictable network latency between different compute nodes. The underlying reason for the performance loss is the under-utilization of hardware resources of the GPU accelerators. In Step 1 and 3, compute units are completely idle waiting the data synchronization. While in Step 2, the DMA units are doing nothing since the compute units are executing the kernel functions. Therefore, overlapping the data synchronization and the kernel execution will reduce the total running time of the training process. To achieve this goal, we break down the replicas into finer grained replicas with the same number of compute nodes used. Then the kernel execution of one replica can run simultaneously with data synchronization of other replicas. Ideally, if the replica break-down does not introduce any communication overhead, the more replicas we have in each node, the better performance we will get. However, the communication overhead is not negligible so we have to find the best number of replicas per node. Since the ratio of communication over computation varies across different types of neural networks, we need to carefully evaluate and design workload mapping methods to achieve the best efficiency on multi-node GPU-enabled machines.

D. Why Blue Waters

The Blue Waters offer XK7 nodes which consist of one AMD 8-core CPU and one NVIDIA K20 GPU. As GPUs are more suitable than CPUs for convolution and matrix multiplications, which are the major computation in deep learning, state-of-the-art deep learning facilities widely employ GPUs as their hardware accelerators. The Blue Waters offers us an opportunity to do research on optimization for deep learning cluster with GPUs. Especially the system provides CUDA programming environment, which enables us to customize the specific functions to be executed on GPUs.

III. Experiment Setup and Results

In this section, we present the evaluation results generated from our experiments on the Blue Waters. To evaluate the performance of different types of neural networks, we choose a popular neural network, AlexNet, for referece of convolutional layer and fully connected layer topologies. AlexNet has one convolution layer of $(224, 3, 11)$, (the numbers are the size of input image, the number of channels and the size of filter kernels), one convolution layer of $(55, 96, 5)$, one convolution layer of $(27, 256, 3)$ and two convolutional layers of $(13, 384, 3)$. The size of the fully connected layers in AlexNet

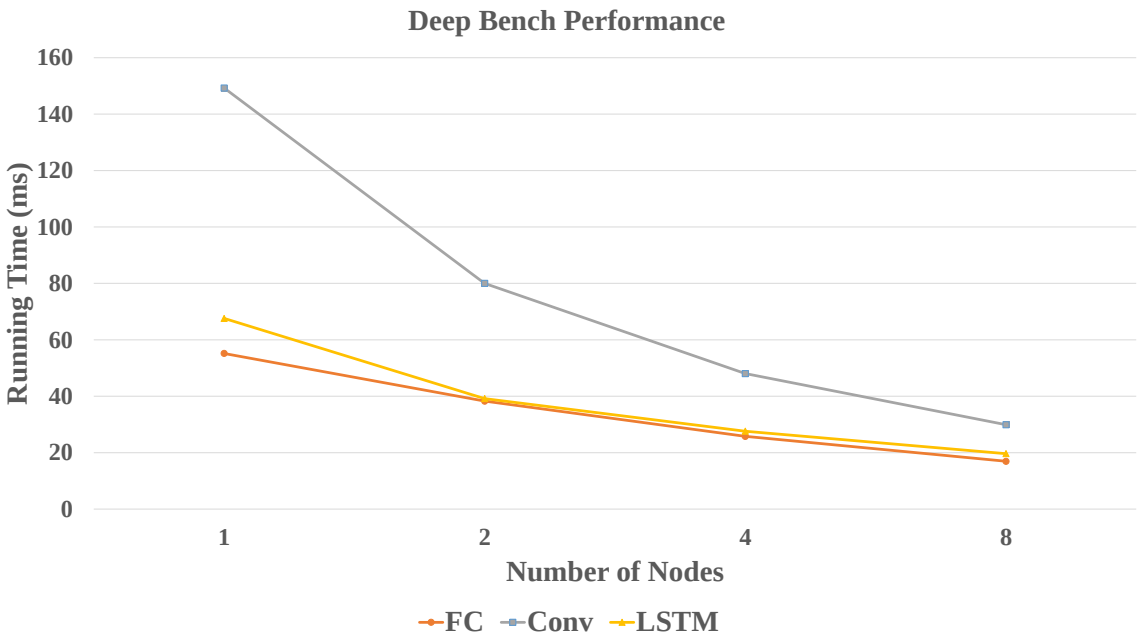

Fig. 2: Performance Evaluation of Different Layer Types

are 4096, 4096 and 1000, respectively. For the topology of LSTM RNNs, we choose a charcter-based language model of which all recurrent layers have 128 neurons. Since all the LSTM layers are the same, we only use one LSTM layer to run the experiment.

We implemented these neural network layers based on the DeepBench[4], which is a performance benchmark for deep learning hardware accelerators. We modified the DeepBench to change the OpenAPI originally used to the platform API of the Blue Waters. All nodes allocated are XK7 GPU enabled nodes.

Figure 2 shows the running time of each type of neural networks on different number of nodes. In the figure we can see that the running time of all the three types of layers reduces along with the increase of the nodes used in parallel. The Blue Waters system shows good scalability, though there is communication overhead which makes the speedup sub-linear. From Figure 2 we all observe that the speedup for different types of neural networks are different since they have different computation per byte. This observation indicates that if we cannot achieve the best performance or system efficiency if we use one single resource allocation scheme for all the three types of neural networks. For example, the communication dominates the latency for LSTM layers and fully connect layers in the case where we allocate 8 nodes, while convolutional layers are still computation-bound. Based on this observation, we will design new resource allocation and algorithm mapping techniques to achieve better system performance given a fixed amout of workload.

IV. Next Generation Work

In future work, we will build a model to find the best number of nodes for each type of neural network layers given a specific size. To verify our model against real-world supercomputer systems, we need to run more experiments on the Blue Waters to compare with the prediction of our model. The experiments will still be based on XK7 nodes which provide GPU support. We expect a conference paper submission after finishing of the next generation work.

References

[1] S. Hochreiter, S. Hochreiter, J. Schmidhuber, and J. Schmidhuber, "Long short-term memory.," *Neural computation*, vol. 9, no. 8, pp. 1735–80, 1997.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[3] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," pp. 1–14, 2015.

[4] "Deepbench." https://github.com/baidu-research/DeepBench. Accessed: 2016-10-30.