used for simulation. In our effort, we developed a synthesis technique for the implementation of equivalent circuits based on scattering parameter representation of linear broadband networks. Several circuit topologies for complex pole residue pairs were defined and tested and found to accurately describe the behavior of the blackbox networks. When the number of ports becomes large, the computational efficiency of the method is very limited, which warrants the use of a platform such as Blue Waters.

WHY BLUE WATERS

We plan to use the Blue Waters computational platform to explore the feasibility in simulating high-speed serial links that would necessitate several thousand years using standard algorithms. By properly combining the computational powers of the Blue Waters system with new simulation algorithms developed in our research group, we plan to reduce the computational time by several orders of magnitude. More specifically, our research will include the following tasks: use LIM on Blue Waters to make transistor-level simulations involving a large numbers of devices (this would not be feasible without Blue Waters); develop the circuit synthesis of blackbox macromodels with large numbers of ports; develop techniques to parse, analyze, and process the big data generated by measurement or calculation of X parameters; and process the generated data to develop deterministic and stochastic simulation methods for analog blocks using Volterra series and the latency insertion method (LIM) on the Blue Waters platform.

NEXT GENERATION WORK

In the 2019-2020 time frame, we plan to demonstrate the simulation of integrated circuits with large numbers of transistors as well as high-speed links.

HARDWARE ACCELERATION OF DEEP LEARNING

Allocation: Illinois/50.0 Knh
PI: Tao Xie[1]
Co-PI: Yuan Xie[2]

[1]University of Illinois at Urbana-Champaign
[2]University of California at Santa Barbara

EXECUTIVE SUMMARY

Our project aims to use Blue Waters for hardware acceleration of deep learning for big data image analytics. To achieve near real-time learning, efforts are required for 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). We evaluated the performance of scaling up using the graphics processing unit (GPU) enabled node XK7 for training convolutional neural networks. Our key observation thus far is that implicit data synchronization across different nodes severely slows down the training process. We propose a data manager that explicitly covers the data transfer overhead with computation. Our first step was to test the proposed strategy on a single XK7 node of Blue Waters, and results show a speedup of 1.6 times that of the implicit data transfer implementation.

INTRODUCTION

Deep learning has been used in applications such as image classification, speech processing, and object recognition. A very large amount of training data is necessary for deep neural networks, therefore this work requires computing power capable of matching the advanced state-of-the-art accuracy of these tasks. Mainstream deep learning facilities are central processing unit (CPU)-based clusters, which usually consist of thousands of compute nodes. As the major computation of deep learning is convolution and matrix multiplication, which are suitable for GPUs to process, most modern deep learning facilities are equipped with GPUs as hardware accelerators. However, straightforward implementation of deep neural networks on GPU-enabled compute nodes will lead to under-utilization of computing resources. In this work, we evaluated the performance of a convolution neural network on a GPU-enabled cluster (XK7 nodes on Blue Water). We observed that the performance bottleneck of the convolutional neural network training is the implicit data synchronization across different nodes, which is used to confirm the correctness of the output of each layer in the neural network that is distributed in different nodes. To alleviate the performance degradation caused by the synchronization, we propose a method to hide the data transfer explicitly by computation. This method dramatically increases the utilization of the compute units on each GPU chip and thus improves the overall training performance.

METHODS & RESULTS

Traditionally distributed convolutional neural networks allocate one compute node for each replica as 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; and 3) sending the output data of the current layer to other replicas.

The performance bottleneck of this straightforward implementation is that steps 1 and 3 introduce very long latency, which is determined by the longest unpredictable network latency between different compute nodes. The reason for the performance loss is under-utilization of hardware resources of the GPU accelerators. In Steps 1 and 3, compute units are completely idle while waiting for data synchronization. In Step 2, the direct memory access units are inactive while 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 combat performance loss due to the under-utilization of hardware resources, we break down the replicas into finer grained replicas with the same number of compute nodes used. This way, 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, there will be more replicas in each node and better performance will be achieved. However, the communication overhead is not negligible, so we have to find the best number of replicas per node. We find the best ratio is two replicas per node, and the speedup of this configuration is 1.6 times over that of straightforward implementation.

WHY BLUE WATERS

Blue Waters offers XK7 nodes which consist of one AMD 8 floating point core CPU and one NVIDIA K20X GPU. As GPUs are more suitable than CPUs for convolution and matrix multiplications, state-of-the-art deep learning facilities widely employ GPUs as their hardware accelerators. Blue Waters offers an opportunity to perform research on equipment optimized for deep learning cluster with GPUs. Also, Blue Waters provides a CUDA API programming environment, which enables us to customize the specific functions to be executed on GPUs.

NEXT GENERATION WORK

A field-programmable gate array (FPGA) is an alternative hardware accelerator to GPUs. It can be quickly used to prototype new hardware designs with very high power efficiency. In the next generation work plan could employ a FPGA to accelerate deep learning algorithms.