Large Scale Artificial Neural Network Training Using Multi-GPUs

Linnan Wang
NEC Labs
linnan.wang@gatech.edu

Wei Wu
University of Tennessee, Knoxvile
wwu12@vols.utk.edu

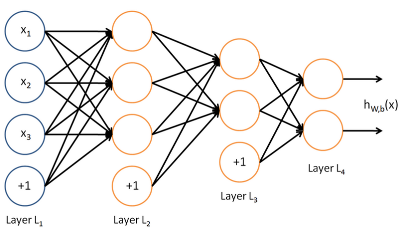
Yi Yang
NEC Labs
yyang@nec-labs.com

Jianxiong Xiao
Princeton University
xj@princeton.edu

Alex Zhang
Rutgers, New Brunswick
alx.zhng@gmail.com

Introduction

1. Artificial Neural Network (ANN) is widely used in a variety of data analysis



- pattern classification
- clustering
- function approximation
- forecasting

Motivation

1. Benefit from large scale

- <u>*Reducing the approximation error*</u>

In the field of function approximation, the integrated squared error of approximation, integrating on a bounded subset of *d* variables, is bounded by *cf* / *n*, where *cf* depends on a norm of the fourier transform of the function being approximated [1]. Therefore, the approximation error can be reduced by increasing the network size.

- <u>*Increasing the model capabilities*</u>

In the field of natural language processing, the output nodes in a neural network are usually required to be equal to the size of vocabulary [2], which often exceeds 10^4. A larger network enables itself to recognize bigger vocabulary.

- <u>*Improving the accuracy of forecasting*</u>

In the field of forecasting, the number of input nodes in an ANN tie to the number of variables in a predicting model. A larger network can improve the forecasting accuracy by incorporating more variables into the model.

2. The challenge of training a large scale ANN

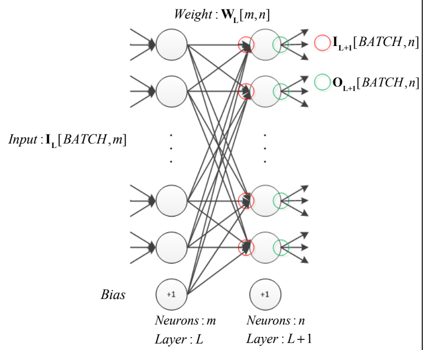
- <u>*Large scale ANN training is computation intensive*</u>

For a network with 10^4 output nodes and 10^3 hidden nodes, it has at least 10^7 tuning parameters. Training of such a network may take hundreds of hours on a CPU.

- <u>*Large scale ANN training is parameter intensive*</u>

Although a GPU can accelerate the training phase 10x faster, the ANN parameters have to fit into the fixed and limited GPU memory.

Reducing ANN Training to Matrix Multiplication



1. Forward pass

$$I_{L+1}[BATCH,n] = O_L[BATCH,m]\cdot W_L[m,n] + \mathbf{Bias}_L[BATCH,n] \quad [1]$$

$$O_{L+1}[BATCH,n] = SIGMOID(I_{L+1}[BATCH,n]) \quad [2]$$

2. Backward pass

$$\frac{dE}{dW_L}[m,n] = O_L^T[m,BATCH]\cdot \frac{dE}{dI_{L+1}}[BATCH,n] \quad [3]$$

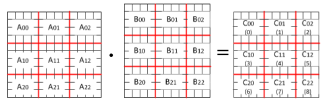
$$\frac{dE}{dO_L}[BATCH,m] = \frac{dE}{dI_{L+1}}[BATCH,n]\cdot W_L[n,m] \quad [4]$$

<u>Large batch and huge numbers of neurons</u>

<u>Giant matrix multiplication</u>

A Tiled Version of Multi-GPU Matrix Multiplication with Dynamic Task Scheduling

1. Matrices in tile representation



The tile algorithms treat a matrix as a set of matrices, and the matrix multiplication in tiled version is:

$$C_{ij} = \alpha\sum_k A_{ik}B_{kj} + \beta C_{ij}$$

Tile indices *i*, *j*, *k* govern the multiplication.

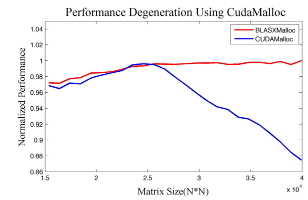
2. Defining tasks

We define solving C_{ij} as a task.

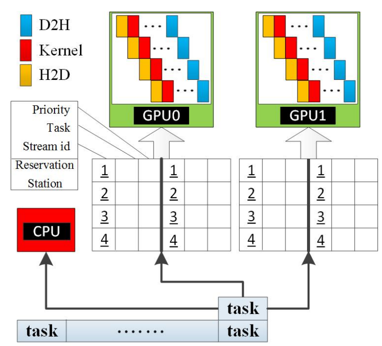
The following are three notable properties of a task:

- <u>*Reading the input of tasks is data dependency free.*</u>
- <u>*Concurrent writing the result of tasks is data race free.*</u>
- <u>*The workload of each task varies.*</u>

3. Amortizing high frequency memory allocation and deallocation

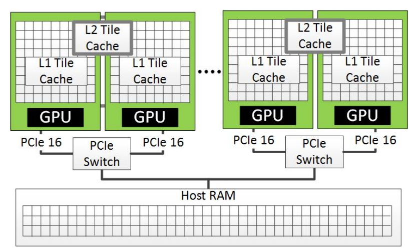


4. A simplified dynamic task scheduling runtime for matrix multiplication



Non-blocking task queue allowing efficient concurrent queue operations

5. Tile reuse by multi-GPU cache coherence

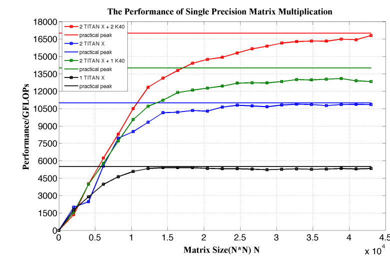


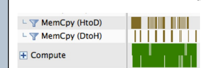
- <u>*A tile is on the private GPU memory => use the tile directly.*</u>
- <u>*A tile is not on the private GPU memory but available on other GPUs => retrieve the tile from the closest GPU according to the hardware proximity.*</u>
- <u>*A tile is not available on all the GPUs => retrieve the tile from the host memory*</u>

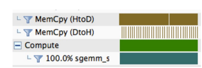
Experimental Setup

	System Configurations
GPU	2 NVIDIA TESLA K40c + 2 NVIDIA TITAN X
CPU	Xeon E5 2637 V3
OS	Ubuntu Server 14.04.1
RAM	128 GB DDR3
CUDA	V 6.5
Compiler	GCC 4.8
CPU BLAS	OpenBLAS v 1.13

Experimental Results







The profile with cache coherence   Without cache coherence

Integration with ANN Training:

Input Data: CIFAR10 dataset and the batch size is 16384

ANN Setup1:

Input Layer	3072 (32*32*3)
Hidden Layer 1	16384
Hidden Layer 2	16384
Output Layer	10

Performance:

Computing Device	Forward and Backward Time (ms)	Speed Up W.R.T Caffe's CPU	Speed Up W.R.T Caffe's in-core GPU
Caffe's CPU	386142	1	0.048
Caffe's in-core training on 1 TITAN-X	18722	20.6x	1
Our 2 K40c + 2TITAN X Out-of-core	7555	51.1x	2.48

ANN Setup2:

Input Layer	3072 (32*32*3)
Hidden Layer 1	32768
Hidden Layer 2	32768
Output Layer	10

Performance:

Computing Device	Forward and Backward Time (ms)	Speed Up W.R.T Caffe's CPU	Speed Up W.R.T Caffe's in-core GPU
Caffe's CPU	2367836	1	N/A
Caffe's in-core training on 1 TITAN-X	N/A (not enough memory)	N/A	N/A
Our 2 K40c + 2TITAN X Out-of-core	38007	62.3x	N/A

citation

[1] Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *Information Theory, IEEE Transactions on*, *39*(3), 930-945.

[2] Li, B., Zhou, E., Huang, B., Duan, J., Wang, Y., Xu, N., ... & Yang, H. (2014, July). Large scale recurrent neural network on gpu. In *Neural Networks (IJCNN), 2014 International Joint Conference on* (pp. 4062-4069). IEEE.