DNN for Application predictor | Performance Evaluation of CPU vs GPU

Himanshu Jain
Department of Electrical and Computer Engineering
University of Arizona
Arizona, Az -85719
himanshujain@email.arizona.edu

Abstract

This work studies the implementation of deep neural network modeled application predictor based on L1 cache feature on different architectural platforms. The goal of the paper is to discuss the total execution time performance on CPU and clustered based GPU. Using different configuration of L1 Cache, the paper tries to develop a model from data extraction phase to compare the performance of DNN (to find 3300 unknown parameters) on HPC clustered GPU system & local CPU system for a very high dimensional data.

1 Introduction

Advancement of science in AI with the generation of large datasets need more emphasis on the better selection of hardware and software framework. Training of large deep neural network on CPU can be time-consuming step but, HPC ability to take advantages of the computational horsepower of GPUs to deliver real-time performance on large datasets by distributing visualization workloads across a GPU-accelerated cluster with Inherent parallel nature of DNN make the training process on GPU framework many times faster. This experimental work includes the comparison of the execution time of DNN implementation on GPUs and CPU using tensor-flow framework. The work provides the following contribution:

- Optimized python script for data generation and feature selection on gem5 computer architecture tool.
- Optimized python script for feature selection and normalizing the features within each benchmark to efficiently compute and visualize in the 11-dimension dataset.
- Training the labeled dataset using Tensor-flow software framework on different hardware platforms: GPU (UA HPC) & CPU (local).
- Running the bash script and evaluate the performance measures in terms of training & testing loss, accuracy vs time.

The rest of the paper is organized as follows. Section 2 describes the background work related to framework section. Section 3 describes implementation overview as methodology and experimental setup. Experimental analysis is measured in section 4 while section 5 summarizes the work's result & prospective conclusion part.

2 Background

The combination of large data sets, high-performance computational capabilities, and evolving and improving algorithms has enabled many successful applications which were previously difficult or impossible to consider. High-performance CPUs nodes typically support large memory and storage capacities and robust network connectivity. They are more general purpose than GPUs but fail to match them in raw compute capabilities. However, they are typically the best solution for MI applications involving large-scale data analytics where large memory capacity and network capabilities are required.

With the addition of HBM stacked memory to the latest GPU models, the bandwidth is improved to the point where many of the primitives can effectively exploit all available GPU compute resources. For dense linear algebra operations, the performance improvement of GPUs over CPUs is typically in the range of 10-20:1[1]. Sifei Liu[2], uses the DNN model to understand the neural network's operation of spatial prolongation affinities purely using data, rather than hand-designed models. The implemented programming model found to be up to 100x faster than previously possible on GPUs.

Aside from the underlying hardware approaches, a unified software environment is necessary to provide a clean interface to the application layer. Higher-level frameworks (e.g., TensorFlow, Theano) can effectively hide most heterogeneity from application developers as well as enable portability across different systems. Several software frameworks try to optimize different aspects of training or deployment of a deep learning algorithm.

Soheil Bahrampour[3] et. al found that TensorFlow is a very flexible framework, especially in employing homogeneous/heterogeneous devices for the various parts of the computational graph. However, its performance on a single GPU is not as competitive compared to the other studied frameworks. On contrast, John Lawrence [4] et. al worked in his study about Cloud GPU instances in Google Cloud. The paper concludes that the Cloud GPU were not as performant as local PC GPUs even though the GPU in the local PC was older model. Additionally, proved that adding multiple GPUs to the same cloud instance does not make a difference unless one specific program for each new additional GPU.

3 Implementation

3.1 Methodology

The framework presented in Fig1 can be divided majorly into three phases. The first phase describes the process of data generation and normalization factor usage. The data generation phase consists the running of 12 benchmarks of SPEC2016 on gem5 ARM architecture which takes binaries files of benchmark with user-defined configurable parameter. In this, I use 18 configurations for each benchmark producing 216 data sample in total with output of 12 labelled classes. After randomizing, around 20% data is considered for testing phase and rest is for training the model. Normalized parameter within each class is considered so as to scale down each feature within range of 0 to 1. Based on prior experience, I am considering only 11 memory features, 4 each of Instruction and data L1 cache; 1 total cpu_ipc, one for control-0-memory energy and one for control-1-memory energy. Since each data sample is of 11 dimension, the same normalized value for a given parameter of two different classes or configuration will be different in high dimension.

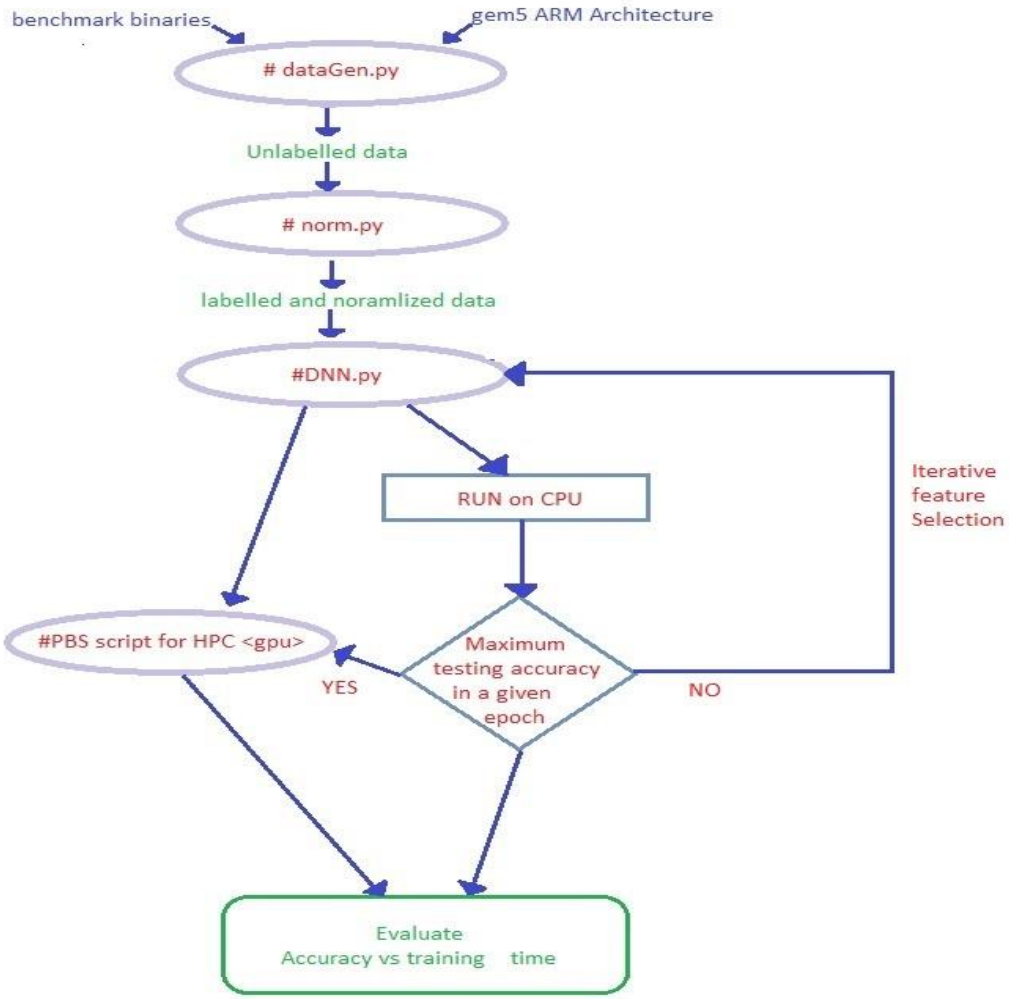

Fig. 1 Flow Diagram of Experimentation

The Second phase includes the development of Deep neural network models using numpy and tensor-flow software libraries in python3.6 with given hyper-parameters as shown in Table 1.

The model is trained on CPU to iteratively select the subset of optimized features on the basis of maximum testing accuracy for a given iteration of training model. After successfully deriving optimized feature, the model is trained on the GPU on UA HPC framework to evaluate and compare the timing to train on different hardware framework, as a final phase.

3.2 Experimental setup

Tensor-flow relies on GPU-accelerated libraries such as CUDNN and NCCL to deliver high-performance multi-GPU accelerated training. The container framework, developed by Nvidia Developers, eliminates the need to manage packages and dependencies or build deep learning frameworks from source.[5]

SPEC2006 benchmark is run on gem5 for different 18 L1 homogenous cache (Instruction and Data Cache have the same configuration). The configuration varied as: line size (in KB) 16,32,64; Associativity: 1,2,4; bank size (in KB): 8,16,32 with some constraint to limit

possible combinations to 18 rather than 27. The resulted data is then normalized for most significant feature extraction to evaluate in 11-dimension space. In this supervised learning, we didn't apply any PCA as it will cause a loss of data. Rather than I took some subset of features, based on prior knowledge, to rank them on training accuracy instead of feature selection tool. ECE3 server was used for the data generation mode.

@ Total Number of features: 11 @ Hidden layers: 3

@ Total Number of Classes: 12 @ Hidden nodes per layer : 100

@ Batch Size: 12 @ Total number of Training samples: 180

@ epoch : 500 @ Total number of Testing Samples: 36

@ Learning Rate: 0.01 @ Total number of Unknown parameters: 3300

For CPU evaluation, model is trained on personal laptop using pip command to install tensor-flow libraries. The CPU configuration consists of intel x86 architecture i5 generation with 4 core single-threaded processor, having 1.8 GHz speed with 8gb DRAM.

For GPU evaluation, UA HPC was configured using PBS script. The user is allocated with 28 CPUs with 168gb memory with 1 GPU core configuring as Tesla P100-PCIE-16GB, compute capability as 6.0. The OCELOTE node of HPC system uses singularity container to dock Nvidia tensor flow frame-work nvidia-tensorflow.18.03-py3.simg, presented in HPC at singularity container directory.

4 Experimental Analysis

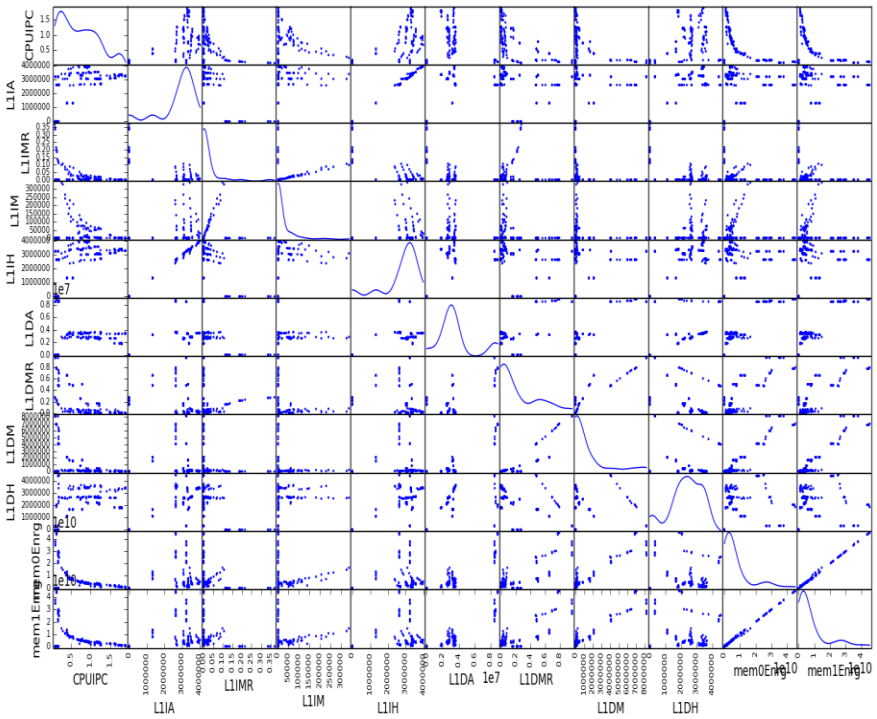

Fig 2 Pairwise Dependency of features of raw data on each other

We infer from the Fig 2. that memory ctrl energy feature is inversely proportional to IPC while memory ctrl0&1 energy are linearly dependent. Other features dependency can't be inferred directly.

The iterative phase of feature section chooses only given subset of features as described in table 1. By plotting the loss and Accuracy for train and testing [Fig 3 & Fig 4], we found that features set included in iteration 5 are prominent to achieve fast and good testing accuracy value in a given epoch.

Iteration Number	IPC	IC Access	IC miss rate	IC misses	IC hit	DC Access	DC miss rate	DC misses	DC hit	Ctrl0 mem enrg	Ctrl1 mem enrg
1	*	*	*			*	*				
2	*	*	*	*		*	*	*			
3	*	*	*		*	*	*		*		
4	*	*	*	*	*	*	*	*	*		
5	*	*	*	*	*	*	*	*	*	*	*

Table 1 Subset of Feature selection during each iteration

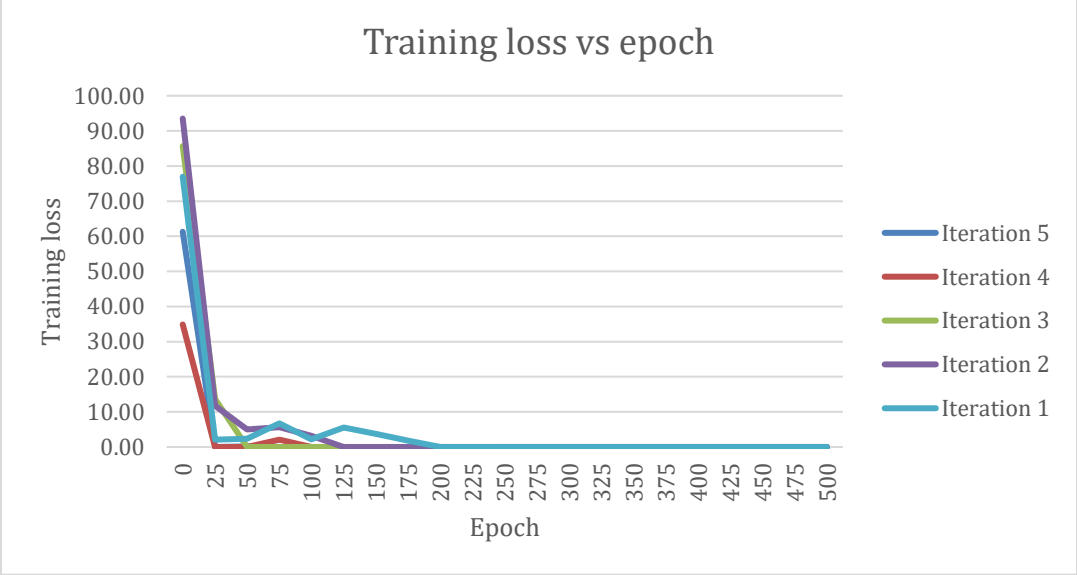

Fig 3 describes the training loss for different features sets used in a different iteration.

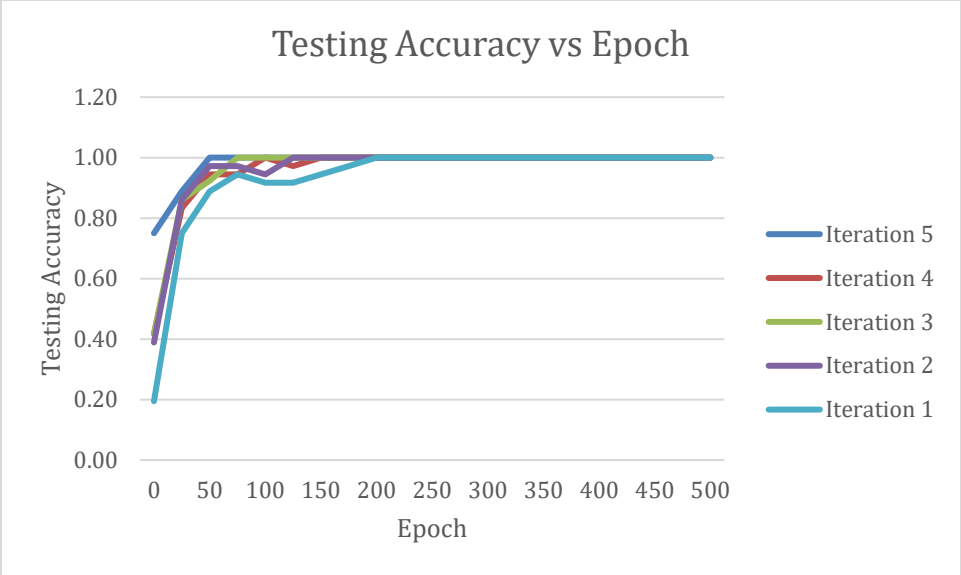

Fig 4 describes the Testing Accuracy for different features sets used in a different iteration.

We use these features to run our final model on CPU vs GPU to find the execution time comparison. Fig 5 & 6 shows the result of Net execution time for on GPU and CPU. From the figure 5 &6, it can be easily found that for small data sample GPU takes more time to train the model due to memory swapping between CPU and GPU. But as the data samples increase or we can say the unknown parameters increases the computability of GPU overcomes the CPU and memory swapping in GPU is of a small part of total execution time. Hence GPU execution time decreases sharply after a given number of sample. One important point is to be noted that since training takes most of the part of the net time & testing phase

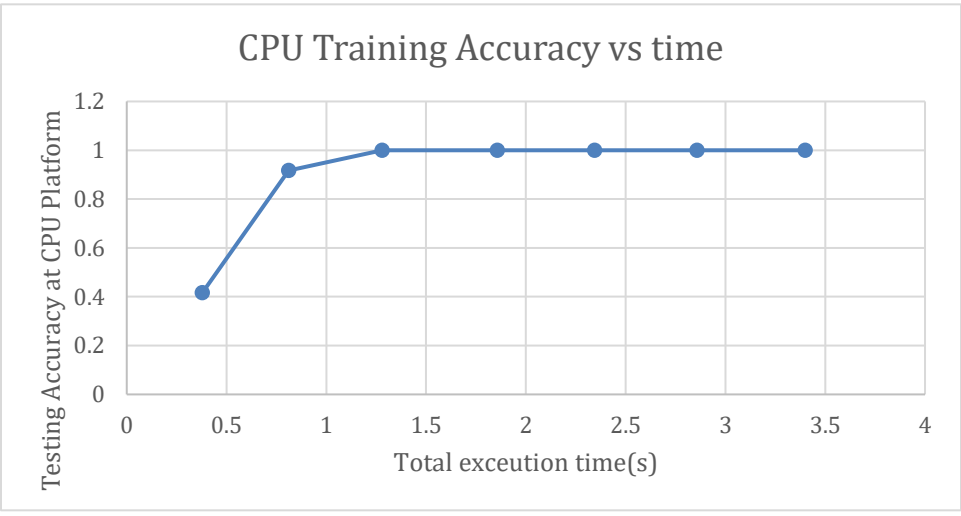

Fig 5 describes the testing Accuracy vs time performed @ CPU

Consists only a small portion as we have already the unknown parameters, so training can be done on GPU and testing can be completed on CPU. This preserves out lot of resources that is used more in GPU than CPU.

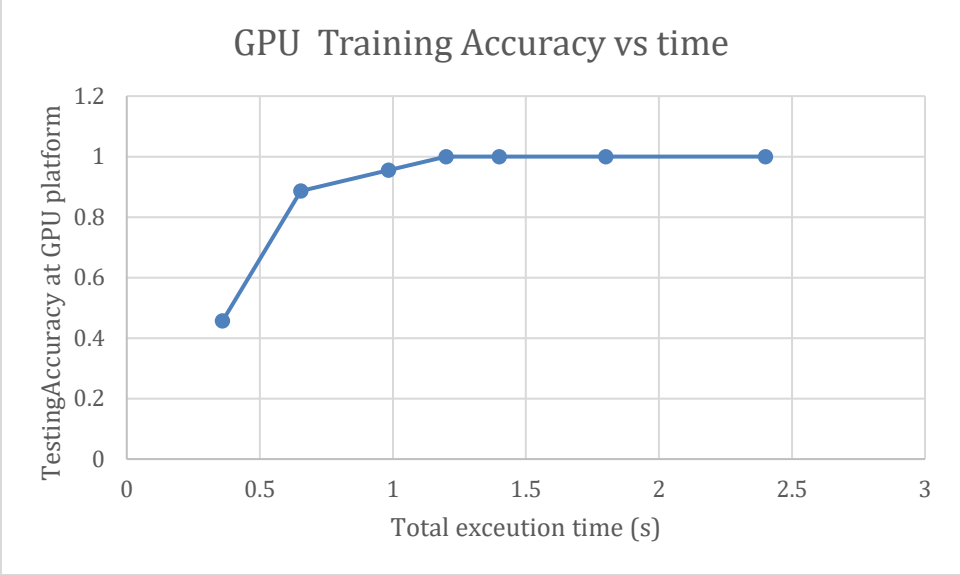

Fig 6 describes the testing Accuracy vs time performed @ GPU

We use these features to run our final model on CPU vs GPU to find the execution time comparison. Fig 5 shows the result of Net execution time for on GPU and CPU. From the figure 5, it can be easily found that for small data sample GPU takes more time to train the model due to memory swapping between CPU and GPU. But as the data samples increase or we can say the unknown parameters increases the computability of GPU overcomes the CPU and memory swapping in GPU is of a small part of total execution time. Hence GPU execution time decreases sharply after a given number of sample. One important point is to be noted that since training takes most of the part of the net time & testing phase consists only a small portion as we have already the unknown parameters, so training can be done on GPU and testing can be completed on CPU. This preserves out lot of resources that are used more in GPU than CPU.

5 Conclusion

In summary, I have developed a DNN model for application predictor using L1 cache features. After data generation from scratch and implementation of normalization concept to evaluate high dimension data with more efficiency, the model training accuracy is compared to CPU and GPU platform. The experiments found in favor of GPU clustered for a given epoch and accuracy. In doing experiment, it is assumed that the subset which is optimized for CPU architecture also performed well over the GPU architecture except for the total execution time.

References :

[1] Allen Rush, Ashish Sirasao, Mike Ignatowski: Unified Deep Learning with CPU, GPU and FPGA technologies. https://pro.radeon.com/_downloads/Unified-Deep-Learning-White-Paper.pdf

[2] Sifei Liu, Shalini De Mello, Jinwei Gu, Guangyu Zhong, Ming-Hsuan Yang, Jan Kautz Learning Affinity via Spatial Propagation Networks. arXiv:1710.01020

[3] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, Mohak Shah: Comparative Study of Deep Learning Software Frameworks. arXiv:1511.06435v3

[4] John Lawrence, Jonas Malmsten, Andrey Rybka, Daniel A. Sabol, and Ken Triplin : Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud. Proceedings of Student-Faculty Research Day, CSIS, Pace University, May 5, 2017.

[5] https://docs.nvidia.com/

[6] https://docs.hpc.arizona.edu/display/UAHPC/GPU+Nodes

[7] https://www.tensorflow.org/