A study of Machine Learning Workloads on different hardware and Benchmarks for deep learning

Sayali S Pangre, Aditya K Sinha, Milind Bhandare

Abstract -- Deep learning is a very popular machine learning method currently, and it can be used to solve various tasks. There are now many open source deep learning tools that can build various deep learning network models, such as convolutional neural network (CNN) and recurrent neural network (RNN), Generative Adversarial Networks(GANs). However, deep learning workloads are becoming increasingly more compute-intensive, so training deep learning networks is usually a very time-consuming process. Almost all deep learning frameworks support the use of GPU in order to speed up the calculation of deep learning models. This dissertation will involve benchmark to analyze the performance of CPU and GPU in training deep learning workloads on PARAM Shavak Deep Learning GPU. ParaDnn is a micro-benchmark for deep learning, which can compare the running time of the workloads between various devices. In addition, a simple deep learning model is also built to compare the running time between one GPU and multiple GPUs. The results of this paper demonstrate that running workloads on GPU is faster than CPU, and multiple GPUs are faster than one GPU.

I. INTRODUCTION

Over the last ten years, deep learning has been successfully applied in various application fields such as computer vision, image classification, speech recognition and natural language processing, etc [1]. At the same time, as the amount of data increases, a large amount of calculations required for training a deep learning model, which often consumes many days or even months. Therefore, many methods are applied to optimize their computing performance. With the rapid development of GPU from a configurable graphics processor to a programmable parallel processor, programs are increasingly using the massive parallel computing capabilities of GPUs to achieve superior performance and efficiency [2]. Today, GPU computing makes it possible for applications that we previously thought were impossible to achieve due to the long execution time [2]. We hope to build deep learning models and run them on the CPU and GPU to obtain running time and then analyze the performance of the deep learning workloads between various devices by using a benchmark for deep learning.

The goal of this research is to choose an appropriate benchmark to analyze the performance between various devices in training deep learning workloads. The running time of deep learning models in different devices can be used to compare the computing performance of CPU and GPU.

In this paper, we focus on analyzing the running time performance of FC, RNN and CNN models. This paper could be divided into two tasks: 1. Construction of ParaDnn benchmark – ParaDnn is a tool that can generate parameterized end-to-end models to run on target platforms [3]. Fully connected networks (FC) and recurrent neural network were generated by ParaDnn [3]. TensorFlow framework was used in this task to build these deep learning models. 2. Construction of convolutional neural network based on Keras and Tensorflow– A convolutional neural network was built in this task based on Keras, and the dataset for training the model is MNIST (Mixed National Institute of Standards and Technology database), which is comprised of a training set of 60,000 images and a test set of 10,000 images.

The main contribution of this dissertation is the study of different benchmarks and analyzing performance of CPU and GPU, Param Shavak DL GPU in training deep learning models. In this paper, various deep learning models was developed, and they were run on CPU and GPU respectively and check their performances. This study could provide a simple model of deep learning to compare the computing performance of CPU and GPU.

II. BACKGROUND AND RELATED WORK

A. Architecture of GPU

NVIDIA is a well-known company that designs and manufactures different GPUs. In 2007, NVIDIA released a programming technique, which consists of a programming model named Compute Unified Device Architecture (CUDA) and a compiler that supports the C language with GPU specific extensions for local, shared and global memory, texture memory, and multithreaded programming [4]. As shown in Figure 1, a GPU is an array of streaming multiprocessors (SMs), and each has a number of streaming processors (SPs). One GPU device contains multiple SMs. GPU can perform more tasks at the same time if there are more SMs in the GPU. These SMs are first connected to a shared memory (L1 cache), and then they are connected to an L2 cache. Each SM needs to access a register file, which is a set of storage units that can work at the same speed as SP, so accessing this set of storage units requires almost no waiting time [5]. In addition, there is a shared memory that is only accessible internally by each SM, which can be used as a cache for program management [5]. For texture memory, constant memory and global memory, each SM has a bus that can access them independently [5]. Texture memory is a special view of global memory that is useful for data that has interpolation, such as using 2D or 3D lookup tables [5]. It has the ability to interpolate based on hardware. Constant memory is used to store read-only data, which is cached by all GPU cards [5]. Like texture memory, constant memory is also a view of global memory [5].

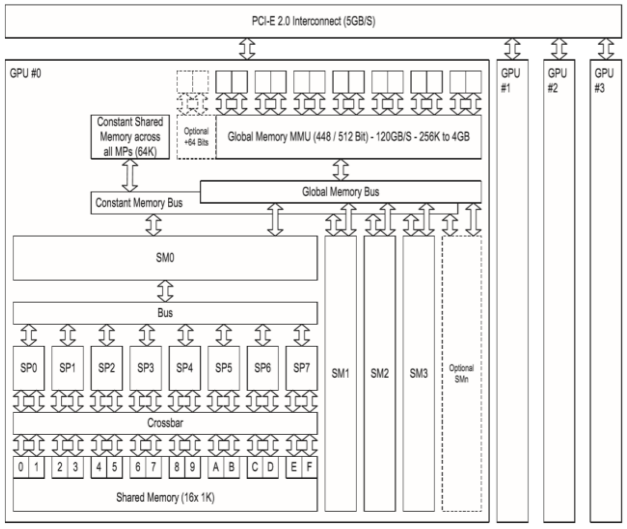

Fig.1: Block diagram of a GPU card

B. GPU Computing Capability

As given in [6], GPUs have higher computational power than CPUs, and the divergence of CPU and GPU computational power is becoming larger. The main reason of the divergence between CPU and GPU is fundamental architectural differences [6]. Data parallel in graphics computing allows GPUs to use additional transistors for calculations more directly, achieving higher arithmetic intensity with the same number of transistor count [6]. Thus, GPUs are widely used in accelerating computing for complicated projects. There are many deep learning frameworks that can run on both CPU and GPU, such as TensorFlow, PyTorch and Keras, Caffe2, MxNet. In this study, Param Shavak DL GPU was used to accelerate deep learning workloads based on TensorFlow and Keras.

C. Benchmarks Survey

1) DeepBench

The primary purpose of DeepBench is to benchmark operations that are important to deep learning on different hardware platforms. Although the fundamental computations behind deep learning are well understood, the way they are used in practice can be surprisingly diverse. For example, a matrix multiplication may be compute-bound, bandwidth-bound, or occupancy-bound, based on the size of the matrices being multiplied and the kernel implementation. Because every deep learning model uses these operations with different parameters, the optimization space for hardware and software targeting deep learning is large and under specified[15].

2) tf_cnn_benchmarks

tf_cnn_benchmarks contains TensorFlow 1 implementations of several popular convolutional models, and is designed to be as fast as possible. tf_cnn_benchmarks supports both running on a single machine or running in distributed mode across multiple hosts[26].

3) DawnBench

DAWNBench is a benchmark suite for end-to-end deep learning training and inference. Computation time and cost are critical resources in building deep models, yet many existing benchmarks focus solely on model accuracy. DAWNBench provides a reference set of common deep learning workloads for quantifying training time, training cost, inference latency, and inference cost across different optimization strategies, model architectures, software frameworks, clouds, and hardware. Building on our experience with DAWNBench, we helped create MLPerf as an industry-standard for measuring machine learning system performance[16].

4) ParaDNN

ParaDnn is a tool that generates parameterized deep neural network models. It provides large “end-to-end” models covering current and future applications, and parameterizing the models to explore a much larger design space of DNN model attributes[32].

5) AI Bench

AIBench adopts a scenario-distilling AI Benchmarking methodology. Instead of using real-world applications, we propose the permutations of essential AI and non-AI tasks as a scenario-distilling benchmark (in short, scenario benchmark). Each scenario benchmark is a distillation of the essential attributes of an industry-scale application, and hence reduces the side effect of the latter’s complexity in terms of huge code size, extreme deployment scale, and complex execution paths. We consider scenario, component and micro benchmarks as three indispensable parts of a benchmark suite[21][31].

6) MLPerf

MLPerf was built for fair and useful benchmarks for measuring training and inference performance of ML hardware, software, and services[22].

7) Fathom

Fathom is a collection of eight archetypal deep learning workloads for study. Each of these models comes from a seminal work in the deep learning community, ranging from the familiar deep convolutional neural network of Krizhevsky et al., to the more exotic memory networks from Facebook's AI research group[23].

8) BigDataBench

BigDataBench 5.0 provides 13 representative real-world data sets and 27 big data benchmarks. The benchmarks cover six workload types including online services, offline analytics, graph analytics, data warehouse, NoSQL, and streaming from three important application domains, Internet services (including search engines, social networks, e-commerce), recognition sciences, and medical sciences. Our benchmark suite includes micro benchmarks, each of which is a single data motif, components benchmarks, which consist of the data motif combinations, and end-to-end application benchmarks, which are the combinations of component benchmarks[17],[24].

9) PerfZero

PerfZero is a benchmark framework for TensorFlow. It intends to address the following use-cases - For users who want to execute TensorFlow test to debug performance regression.
PerfZero makes it easy to execute the predefined test by consolidating the docker image build, GPU driver installation, TensorFlow installation, benchmark library checkout, data download, system statistics collection, benchmark metrics collection, profiler data collection and so on into 2 to 3 commands. This allows developer to focus on investigating the issue rather than setting up the test environment - For user who wants to track the performance change of TensorFlow for a variety of setup (e.g. GPU model, cudnn version, TensorFlow version)[22]

10) DLBS (HP)

Deep Learning Benchmarking Suite (DLBS) is a collection of command line tools for running consistent and reproducible deep learning benchmark experiments on various hardware/software platforms. Deep Learning Benchmarking Suite was tested on various servers with Ubuntu / RedHat / CentOS operating systems

with and without NVIDIA GPUs. We have a little success with running DLBS on top of AMD GPUs, but this is mostly untested. It may not work with Mac OS due to slightly different command line API of some of the tools we use (like, for instance, sed) - we will fix this in one of the next releases[25].

11) Training Benchmark Suite(TBD)

TBD is a Benchmarking Suite for DNN training which covers six major applications by performing extensive performance analysis of training these different applications on three major deep learning frameworks (TensorFlow, MXNet, CNTK) across different hardware configurations (single-GPU, multi-GPU, and multi-machine)[29].

12) Intel MPI Benchmark

This Benchmark is used to compare and evaluate the combined performance of processor, memory subsystem and interconnect fabric[30].

13) BenchIP and BenchNN

Use to design intelligence processors architecture and neural network workloads[27],[28].

D. Training and Inference

Training a DNN requires a very large labelled data set and network architecture that will learn features and model. This might take hours to weeks depending on the dataset, computational power and algorithms that are used for training. The trained neural network uses what it has learned to recognize images, spoken words, or suggest new stuff someone is likely to buy next. This faster and efficient version of a neural network infers things about new data it is presented based on its training.

In inference it uses this learn knowledge for most of the part. Inference does not require all the infrastructure of its training to do its task well. While training DNN , training data is put into the input layer of the network, and individual neurons assign a weighting to the input based on the task being performed. The term deep refers to number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.

In an image identification network, the first layer might look for edges. The second might look for how these edges form shapes. The third might look for particular features and so on. Each layer passes the image to the next, until the final layer and the final output determined by the total of all those weightings is produced.

Many deep learning applications use the transfer learning approach, which uses pre-trained model, where it starts with an existing network, such as AlexNet, VGG16, Mask R-CNN or GoogleNet, and give in new data containing previously unknown classes. After making some adjustments to the network, you can now perform a new task, such as classifying only dogs or cats instead of 1000 different objects. This gives the advantage of needing much less data processing thousands of images, rather than millions, so computation time drops to minutes or hours.

*1) **How inference works?***

First approach it looks for the parts that are not activated after it is trained. These parts not needed and can be pruned away. The **second approach** looks for ways to fuse multiple layers of the neural network into a single computational step, which means we use inference all the time. For example Google, Facebook, Baidu, Amazon and Netflix uses inference for different applications.

GPUs have the parallel computing capabilities I.e. the ability to do multiple things at once are good at both training and inference. After training the models are deploy for classifying data to infer a result. Here as well, GPUs are used, where they run billions of computations based on the trained network to identify known patterns or objects.

E. CPU vs GPU General Performance

It is clear that architectural performance of CPU and GPU provide different performance Sometime CPU gives better performance than GPU and sometimes GPU gives better performance than CPU ,however better here is relative to the application CPU and GPU will be used and they both function very differently.

CPU's are powerful and that is the reason computer can do any task and they are more flexible than GPUs, they include a larger instruction set, they run at higher clock speeds, perform IO operations and also they are responsible for integration with virtual memory which GPU cannot do. CPU's perform their tasks sequentially I.e. one task at a time.

Contrary, GPUs are optimized to display graphics and to do specific computational tasks. GPUs perform fraction of the operations with a very high speed due to its lower clock speeds, which makes it useful for image processing. GPU uses its hundred of cores to perform time sensitive calculations for thousand of pixels at a time, thus be able to display any type of images as well including complex 3D graphics as well because of its architecture to perform multiple parallel operations at time. It needs mathematical operations to be done during processing of images.

F. CPU vs GPU Deep Learning Performance

As given above ,CPU's have larger instruction set than GPUs making GPUs less flexible , however GPUs are said to be dedicated for parallel computing even for same instructions. Deep Neural Networks (DNN) are structured in a very uniform manner such that at each layer of the network thousands of identical artificial neurons perform the same computation. Therefore, its way of computation is quite similar to how GPU computes instructions.

As specified, image processing is an expensive task that requires many calculations and since GPUs have more computational units and have a higher bandwidth to retrieve from memory ,GPUs perform quite well in that task with high speed as well,deep learning includes massive image processing operations with large data sets making parallelism a needed feature in deep learning computing .

The main weakness of GPUs as compared to CPU's is memory capacity I.e. GPUs are lower than CPU's. The highest known GPU contains 24GB of RAM, in contrast, CPU's can reach 1TB of RAM. A second weakness is that a CPU is required to transfer data into the GPU card. CPU helps to feed GPU with enough data and read/write files from/to RAM/HDD during training. If the CPU is weak, it can only feed as few data as possible thus cannot keep up with your powerful GPU. Ideally Deep Learning training systems should have CPU with maximum number of processing cores to handle more work to catch up with a GPU. In addition, GPU clock speeds are 1/3rd that of high end CPU's, so on sequential operations won't be as fast as if they were processed using a CPU.

However, GPUs are so efficient and fast in matrix multiplication and convolution due to parallelism and memory bandwidth. CPU's are latency optimized while GPUs are bandwidth optimized. CPU can fetch small amounts of memory much faster than GPU which can fetch large amount of memory at slower rate. This means larger computational operations, the more advantages of GPUs over CPU's. But, GPUs are not latency optimized which might affect the GPUs performance. However this problem was solved by thread parallelism. By this we can say that GPUs provide best memory bandwidth while having no drawback due to latency when

thread parallelism is used.This is one of the main reasons why GPUs are faster than CPUs for deep learning.

One more benefit of GPUs is that they consist of a small pack of registers for every processing unit ,therefore a lot of register memory which is small and fast, this provides GPUs with registers size more than 30 times bigger compared to CPUs but yet very fast .This difference in size is much more important than difference in speed and it does not make a difference. And a good compiler tools that can exactly indicate when we are using too much or too few registers ,maximal performance is sure guaranteed.

This finally leads to the conclusion that we can store a lot of data on register files on GPUs ,in order to be able to reuse convolutional and matrix multiplication tiles. You have a 100MB matrix, you can split it up in smaller matrices that fit into your cache and registers, and then do matrix multiplication with three matrix tiles at speeds.That is again why GPUs are much faster than CPUs in deep learning which requires a lot of matrix multiplications. To sum up, High bandwidth ,hiding memory access latency under thread parallelism and having large and fast register files that can be easily programmable , makes GPUs more fit when it comes to deep learning.

III. Deep Learning Frameworks

A. TensorFlow

TensorFlow is a popular machine learning system that is widely used for building various neural network models and deep learning models. TensorFlow integrates the most common units in deep learning frameworks and supports many latest networks with different settings, such as CNN and RNN [1]. TensorFlow supports both large scale training and inference and it can effectively use many powerful servers such as GPU for rapid training [7]. TensorFlow source code contains a file named Stream Executor [8], which can call CUDA platform to use GPU.

B. Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow [9]. It was developed with a focus on enabling fast experimentation [9].

C. PyTorch

PyTorch, the biggest competitor for Tensor Flow is developed by Facebook and written in Python, C, and CUDA. PyTorch is based on Torch, an open source machine learning library written in Lua. This Python-first framework integrates Python and allows the use of any Python library to build neural network layers [33].

D. Caffe2

Caffe is deep learning framework developed by the Berkeley AI Research (BAIR) team. Caffe stands for Convolutional Architecture for Fast Feature Embedding. It is written in C++ and has a Python interface as well[34].

E. Microsoft CNTK

The Microsoft Cognitive Toolkit – an open-source DL framework created to deal with big datasets and to support Python, C++, C#, and Java.CNTK facilitates really efficient training for voice, handwriting, and image recognition, and supports both CNNs and RNNs. It is used in Skype, Xbox and Cortana[35].

IV. PARAM Shavak Architecture

The system consists of 2 multicore CPUs each with minimum 12 cores along with two number of accelerator cards. The entire configuration is available in a single server in a table top model. Regardless to the traditional HPC systems/supercomputers, this system does not require specific support infrastructure like precision air-conditioned environment, controlled humidity etc. Also, the accepted sound level is very less when compared to the traditional servers. This brings down the infrastructure cost of the system[12],[14].

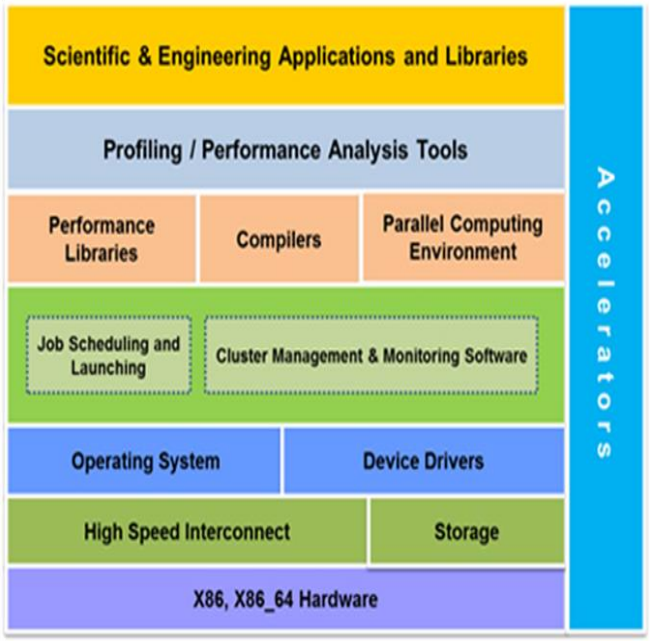

Fig. 2: Block diagram of PARAM Shavak [14]

A. PARAM Shavak DL GPU

C-DAC's Deep Learning development - supercomputer in a box, "PARAM SHAVAK DL GPU System" exclusively designed for academic institutions and research organizations that employ deep learning techniques for GPU accelerated machine learning applications, big data problems (computer vision, speech recognition, natural language processing, life sciences) and artificial intelligence. Equipped with x86 based latest Intel processor, 64 GB RAM, 8 TB storage, NVIDIA Pascal architecture based co-processing technologies (P5000/P6000) and DLGPU software environment (with Deep Learning GPU accelerated libraries and SDK)enabling a new computing platform that's disrupting conventional thinking from the desk-side to the data center. With NVIDIA Pascal architecture inside, the system delivers unprecedented performance up to 25TeraFLOPS of single precision performance for deep learning workloads and enhanced application scalability [14].

V. Benchmark

In order to assess the acceleration effect of GPU on deep learning applications, the benchmark is required. There are many deep learning benchmarks, such as DAWN Bench and MLPerf. DAWNBench is a benchmark that measures end-to-end training time to achieve a state-of-the-art accuracy level, as well as inference time with that accuracy [10]. MLPerf is a benchmark contains sub-items in different fields, including image classification, object recognition, translation, recommendation,

speech recognition, sentiment analysis, and reinforcement learning [11].

A. Methodology

This describes the main method of the project which is in two sections. In the first section, fully connected and recurrent neural networks models are constructed by ParaDnn. We run these two models on CPU and GPU of PARAM Shavak respectively. The second section describes the construction of the convolutional neural network models based on Keras. We run this model on CPU, single GPU and Param Shavak DL GPU respectively, and also run the models on the laptop.

1) ParaDnn

ParaDnn can create fully connected and recurrent neural network models. These models are parameterizable, so ParaDnn models are equal to or greater in size compared to today's real-world models [3].

B. Construction of fully connected models

FC models are comprised of multiple fully-connected layers. The architecture of FC is

Input---> [Layer[Node]] ---> Output,

where [Layer] means the number of layers is variable. We can change the number of layers, the number of nodes per layer, and the numbers input and output units of the data sets.

The neural network can be regarded as a black box that can fit any function. As long as the training data is sufficient, given a specific x, we can get the desired y. Fully connected means that each neuron in the Nth layer is connected to each neuron in the N-1th layer, and each connection has a weight. As shown in the figure 3, there are 2 nodes in the input layer numbered 1 and 2. The hidden layer also has two nodes numbered 3 and 4. The output layer has two nodes numbered 5 and 6, b1 and b2 are bias nodes. w_{ji} represents the weight between the j-th node (unbiased node located at the Nth layer) and the i-th node (unbiased node located at the N-1th layer), where j is the target node and i is the source node. w_{jb} represents the weight between the j-th node (the unbiased node at the Nth layer) and the biased node at the upper layer. a_j represents the output value of the j-th node.

Taking node 3 as an example, the input value of node 3 is
w31x1 + w32x2 + w3b,
and the output value of node 3 is
a3 = σ(w31x1 + w32x2 + w3b),
where σ is the activation function.

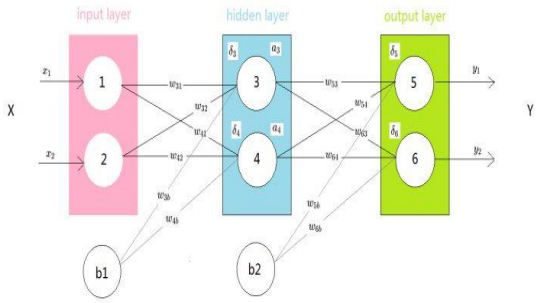

Fig.3: The structure of fully connected neural network

C. Construction of Recurrent Neural Network Models

The fully connected neural network can only take one input after another, and the previous input is completely unrelated to the next input. However, some tasks need to be able to better process the sequence information, that is, the previous input and the subsequent input are related, such as continuous speech and continuous hand written text. RNN is good at this kind of problem. Figure 4 shows the structure of RNN. On the left is the RNN model that is not expanded by time, the right part of the Figure 4 is the RNN model that is expanded by time series. Here, $x^{(t)}$ represents the input of the training sample at the sequence index number t. $h^{(t)}$ represents the hidden state of the model at the sequence index number t, which is determined jointly by $x^{(t)}$ and $h^{(t=1)}$. $o^{(t)}$ represents the output
of the model at the sequence index number t and is only determined by the current hidden state $h^{(t)}$ of the model. $L^{(t)}$ represents the loss function of the model at the sequence index number t. The three matrices U, V and W are the linear relationship parameters of the model, and they are shared throughout the RNN. For any sequence index number t, the hidden state $h^{(t)}$ is obtained from $x^{(t)}$ and $h^{(t=1)}$, that is:
$h^{(t)} = \sigma(z^{(t)}) = \sigma(Ux^{(t)} + Wh^{(t=1)} + b)$,
where σ is the activation function of the RNN, and b is the bias. When the serial index number is t, the expression of the model o(t) is:
$o^{(t)} = Vh^{(t)} + c$
The prediction output for the final sequence index number t is:
$\hat{y}^{(t)} = \sigma(o^{(t)})$
The loss function $L^{(t)}$ such as the log-likelihood loss function can quantify the loss of the model at the current position, that is, the difference between $\hat{y}^{(t)}$ and $y^{(t)}$. In this report,each token of the input sequence is embedded within a fixed length vector, and the length of the vector is the embedding size. We can change the number of layers and the embedding size. We can also change the batch size, which is the number of samples selected in one training. The variables in the dataset include the maximum length per input sequence and vocabulary size.

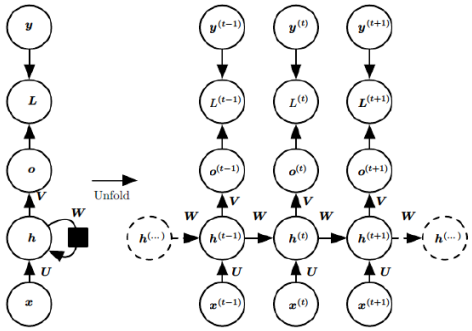

Fig.4: The structure of recurrent neural network[13]

Run this on the CPU and GPU of Param Shavak and also on laptop and write the results to a file to analyze it later. The parameters of models running on the GPU are same as running on the CPU.

VI. CNN AND KERAS

Convolutional neural network is a hierarchical model whose input is raw data, such as images. The convolutional neural network extracts high-level semantic information from the original data through a series of operations such as convolution operation, pooling operation and nonlinear activation function mapping operation. Different types of operations are generally called layers in convolutional neural networks: convolution operations correspond to convolutional layers, and pooling operations correspond to pooling layers. Finally, the last layer of the convolutional neural network formalizes its target tasks (classification or regression) into an objective function. Figure shows the structure of the convolutional neural network.

A. Construction of Convolutional Neural Network Models

This example is a convolutional neural network built by Keras. There are many datasets in Keras that can be used to train CNN. Here we use MNIST dataset, which is a database containing 60,000 handwritten digit pictures that is commonly used for training various image processing systems. We built the following network architecture in our project:

[Convolution]2-[Maxpooling]-[Fully Connected]2

We want to train CNN to implement handwritten digit recognition and classification. First, build two convolutional layers using Conv2D function, which can extract features from 28 ∗ 28 pixels pictures. Next, build a pooling layer by MaxPooling2D to compress the input features, which can simplify network computing complexity. Then, build two fully connected layers using Dense function, which can connect all the features and send the output value to the classifier. When a complete data set passes through the neural network once and returns once, this process is called an epoch. It is not enough to transfer the complete dataset once in the neural network, and we need to pass the complete data set multiple times in the same neural network. In this project, the epochs of the CNN model are 20.

B. Run on CPU and GPU

First, run the model on the CPU of PARAM Shavak. Then, run the model on DL GPU of PARAM Shavak by modifying parameters. Finally, run the model on the laptop, which has an Intel i7-6700HQ CPU and a NVIDIA GTX950M GPU.

VII. Expected Results

This includes the evaluation methods and evaluation results of FC models, RNN models and CNN models. The evaluation method is to compare the running time of the same model on both CPU and GPU.

A. Results of ParaDnn

For fully connected models,
Parameters = (layer + 1) * (node * node + node)
Speedups = (the running time on CPU) / (the running time on GPU)

The speedups of FC models have large ranges, from 1 to 10. The running time of GPU is much less than the running time of CPU. As the complexity of the model increases, the effect of GPU acceleration becomes more significant. For FC models, GPU is a better platform as its architecture can perform large scale parallelism calculations. In addition, FC models rarely reuse weights and large models have more parameters, so they put a lot of pressure on the storage system [3]. The memory bandwidth of the GPU is higher, so running FC models on GPU is faster than CPU.

For RNN,
Parameters = vocabulary size * (2 * embedding + 1) + 601 * embedding * layer
Speedups = (the running time on CPU) / (the running time on GPU)

It is obvious that the running time of GPU is much less than the running time of CPU. Batch size is the number of samples selected in one training. The batch size has a great influence on the speed of models. As the batch sizes of the model increases, the effect of GPU acceleration becomes more significant. This is mainly because increasing the batch size can improve memory utilization through parallelization.

B. Results of Keras

This gives that the GPU has a significant acceleration effect on computing, and the more GPUs used, the shorter the running time. The result of running the CNN model on the laptop with an Intel i7-6700HQ CPU and a NVIDIA GTX950M GPU is the running time of an epoch of the model on the CPU is about 70 seconds, and the running time on the GPU is about 20 seconds. It can be seen that the divergence between the GPU and CPU running time on the laptop is still very large. However, it can be seen that the running time on the GPU of the laptop is a little longer than the running time of the CPU on the Param Shavak DL GPU. This shows that not all GPUs are faster than CPUs because it depends on the computing capability of the hardware. Generally speaking, the computing capability of the GPU on a computer is faster than the computing capability of the CPU on this computer. The hardware equipment of different platforms is different. For example, the hardware performance of the server is much stronger than that of the laptop. Thus, we can use GPUs that have higher computing capabilities to accelerate computation and reduce running time

The GPU is composed of thousands of streaming processors, and single instruction multi-thread (SIMT) mode is used for execution. There are three dimensions in CUDA programming: grids, blocks and threads. A large number of threads form a block, and a large number of blocks form a grid. All threads in each block perform the same operation. If two 1000-element matrices are added together, 1000 threads will be started on the GPU, and one instruction is executed on 1000 threads at a time. Compared with the single-core CPU pipelined serial operation, simultaneous calculation through a large number of threads will obtain a considerable acceleration effect when the amount of data is very large, although the computing power of a single core of the GPU is weak. For CNN, the use of GPU acceleration is mainly in the convolution process, and the convolution process can be parallelized by CUDA like the vector addition. NVIDIA provides the cuDNN library. The deep learning framework implements the core operations of CNN and RNN models by passing tensors and calling the cuDNN library.

VIII. Conclusion and Future Work

This provides a benchmark analysis comparing the performance of GPU and CPU on deep learning workloads. We use ParaDnn and Keras to build up frequently used deep learning models, and we compare the performance of CPU, multiple GPUs. Generally, running workloads on GPU is faster than CPU and use of two GPUs can also reduce the running time compared to only one GPU. Not all GPUs are faster than CPUs, it depends on the device computing capabilities. Also scaling workload on-node GPUs is extremely important. The performance of deep learning algorithms is determined by the structure of the model, using of the perfect hardware and the large available datasets for the training. However, this project still has some limitations. The models we used are too simple, and the datasets are not complicated enough. The number of hardware platforms is not enough. In future, we need to find more benchmarks with more datasets(image or video), and compare the running time, accuracy, latency of inference with different algorithms, systems on more hardware platforms.

REFERENCES

[1] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. In 2016 7th International Conference on Cloud Computing and Big Data (CCBD), pages 99–104. IEEE, 2016

[2] Nickolls, J. and Dally, W.J., 2010. The GPU computing era. IEEE micro, 30(2), pp.56-69.
[3] Wang, Y.E., Wei, G.Y. and Brooks, D., 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. arXiv preprint arXiv:1907.10701.
[4] Yuancheng Luo and Ramani Duraiswami. Canny edge detection on nvidia cuda. In 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, pages 1–8. IEEE, 2008.
[5] Shane Cook. CUDA programming: a developer's guide to parallel computing with GPUs. Newnes, 2012.
[6] John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computa tion on graphics hardware. In Computer graphics forum, volume 26, pages 80–113. Wiley Online Library, 2007.
[7] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016. Tensorflow: A system for large-scale machine learning. In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16) (pp. 265-283).
[8] tensorflow.https://github.com/tensorflow/tensorflow/tree/master/tensorflow/stream_executor/.
[9] Keras: the python deep learning api. https://keras.io/.
[10] Cody Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. Training, 100(101):102, 2017.
[11] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micike vicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, et al. Mlperf training benchmark. arXiv preprint arXiv:1910.01500, 2019.
[12] Agrawal, S., Das, S., Valmiki, M., Wandhekar, S. and Moona, R., 2017, July. A case for param shavak: Ready-to-use and affordable supercomputing solution. In 2017 International Conference on High Performance Computing & Simulation (HPCS) (pp. 396-401). IEEE.
[13] Liu, C., 2020. Analyzing Machine Learning Workloads on Contemporary Processors.
[14] https://www.cdac.in/index.aspx?id=hpc_ss_param_shavak
[15] Baidu DeepBench: Benchmarking Deep Learning Operations on Different Hardware (2018). https://github.com/baidu-research/DeepBench
[16] Stanford DAWNBench: An End-to-End Deep Learning Benchmark and Competition (2018). https://dawn.cs.stanford.edu/benchmark/
[17] BigDataBench: A Big Data and AI Benchmark Suite (2018). http://prof.ict.ac.cn/
[18] Facebook AI Performance Evaluation Platform (2018). https://github.com/facebook/FAI-PEP
[19] MLPerf: A Broad ML Benchmark Suite for Measuring Performance of ML Software Frameworks, ML Hardware Accelerators, and ML Cloud Platforms (2018). https://mlperf.org/
[20] TensorFlow Benchmarks (2018). https://www.tensorflow.org/performance/benchmarks
[21] https://www.benchcouncil.org/AIBench/index.html
[22] https://www.mlperf.org
[23] https://github.com/rdadolf/fathom
[24] https://www.benchcouncil.org/BigDataBench/index.html
[25] https://hewlettpackard.github.io/dlcookbook-dlbs/#/
[26] https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks
[27] Tao, J.H., Du, Z.D., Guo, Q., Lan, H.Y., Zhang, L., Zhou, S.Y., Xu, L.J., Liu, C., Liu, H.F., Tang, S. and Rush, A., 2018. B ench ip: Benchmarking intelligence processors. Journal of Computer Science and Technology, 33(1), pp.1-23.
[28] Chen, T., Chen, Y., Duranton, M., Guo, Q., Hashmi, A., Lipasti, M., Nere, A., Qiu, S., Sebag, M. and Temam, O., 2012, November. BenchNN: On the broad potential application scope of hardware neural network accelerators. In 2012 IEEE International Symposium on Workload Characterization (IISWC) (pp. 36-45). IEEE.
[29] Zhu, H., Akrout, M., Zheng, B., Pelegris, A., Phanishayee, A., Schroeder, B. and Pekhimenko, G., 2018. Tbd: Benchmarking and analyzing deep neural network training. arXiv preprint arXiv:1803.06905.
[30] Bureddy, D., Wang, H., Venkatesh, A., Potluri, S. and Panda, D.K., 2012, September. OMB-GPU: a micro-benchmark suite for evaluating MPI libraries on GPU clusters. In European MPI Users' Group Meeting (pp. 110-120). Springer, Berlin, Heidelberg.
[31] Zhang, Q., Zha, L., Lin, J., Tu, D., Li, M., Liang, F., Wu, R. and Lu, X., 2018, December. A Survey on Deep Learning Benchmarks: Do We Still Need New Ones?. In International Symposium on Benchmarking, Measuring and Optimization (pp. 36-49). Springer, Cham.
[32] Wang, Y.E., Wei, G.Y. and Brooks, D., 2019. Benchmarking TPU, GPU, and CPU platforms for deep learning. arXiv preprint arXiv:1907.10701.
[33] https://pytorch.org/
[34] https://caffe.berkeleyvision.org/
[35] https://docs.microsoft.com/en-us/cognitive-toolkit/setup-cntk-on-your-machine

Sayali S Pangre, Department of Computer Science and Engineering, SOCSE, SandipUniversity, Nashik, India.

Aditya K Sinha, Associate Director HOD-ACTS, CDAC, Pune, India.

Milind Bhandare, Assistant Professor, SandipUniversity, Nashik, India.