# INTROSPECTION INTO DISTRIBUTED DEEP LEARNING ON HPC PLATFORM

Shyamji A. Pandey[1], Dr. Suvanam Sasidhar Babu[2], Gardas Naresh Kumar[3], Samrit Kumar Maity[4]
[1,2]Dept. of Computer Engineering Sandip University, Nashik, India,
[4]HPC Tech, [3,4]CDAC ACTS, Pune, India

***Abstract: Distributed deep learning systems place precise requirement on the communication bandwidth during its model training with wide range of input data processing and training parameter exchange. Usually the communication take place in between cluster of different worker nodes for training data and master parameter node servers for maintaining a global trained model. For immediate convergence the worker nodes and parameter servers(PS) or master node have to frequently exchange huge number of parameters to quickly transmit updates and minimize previous parameters. Hence programming model based on distributed memory Message Passing Interface (MPI) becomes limited to satisfy high bandwidth, low latency, small size but frequent data exchange requirement. Demand on the bandwidth rate gets even higher with the inclusion of dedicated Graphical Processing Units (GPU's) in computing process. Here in this paper we have studied different programming framework suitable for Distributed Deep Learning computation, powered by High Performance Computing (HPC) environment and also aligned with shared memory programming approaches. Recent work on unsupervised feature learning and deep learning has shown that being able to train a large models can dramatically improve the performance. In this paper, we consider different platforms that are involved in implementing high scale distributed network training and parameter sharing. Also virtual memory model based DNN algorithms has been discussed.***

***Keywords: Deep neural network, Distributed deep learning, High performance computing, distributed computing, shared memory, soft memory box.***

## I. INTRODUCTION

High Performance Computing (HPC) is crucial in enabling, technology for the advancement of science and engineering. It supports multi-scale simulations and experiments that leads to breakthroughs discoveries in an ever-broadening range of scientific fields.

In modern Artificial Intelligence (AI) research, Deep Neural Network (DNN) is an approach, introduced to improve machine learning performance even with complicated input data features. The power of DNN has been verified through many applications, especially in visual perception where it showed even better accuracy than human vision system. The success of deep learning in the areas of voice recognition and object recognition is realized because of the availability of massive training data sets, distributed - parallel high performance computing (HPC) architectures and availability of accelerated software framework for the same domain.

General purpose graphic processing unit (GPGPUs) has played an important role in HPC architecture evolution and eventually, acceleration of AI Innovation. Even though it is possible to build more accurate learning models with more training data and large Deep Neural Network models, the computation requirement increases exponentially with the multiplication of model sizes & training data volumes [2] [4].

The success of Deep Learning technology is an effect of three major factors, (i) Evolution of accelerators architecture in the form of GPGPUs [2], (ii) availability of large-scale real time datasets and (iii) invention of various Deep Learning algorithms like back-propagation [2]. Training a Deep Neural Networks (DNNs) is compute intensive and time consuming process. It can take many days or weeks, even on the powerful hardware with GPUs. In pursuit of fast training performance, researchers have resorted to massive parallel GPU devices to conduct network training on a single node over the past few years [4][5].

In deep learning training, it has been observed that increasing the size of deep learning models in terms of number of training examples and the number of model parameters can drastically improve classification accuracy of the particular model with respect to datasets and previous results [1]. To achieve that goal, one prominent approach is to train the model on large distributed memory system like HPC clusters. There

are many distributed DL framework [1, 2, 3]. In distributed DNN training, each worker share parameters located on parameter server (PS). After minute modification in gradients, each worker sends updated gradients to PS, receives the updated parameters from parameter server repeatedly until the training process is completed. This communication pattern generates a large amount of network traffic. The larger network models are, more communication between PS and workers per iteration. This causes massive communication overhead in the whole process. This overhead is a major performance bottleneck in distributed Deep Neural Network training. Therefore, performing efficient parameter sharing has become a worthy goal to chase in distributed DL.

There are two main strategies for parallelizing distributed training method, namely data parallelism and model parallelism [3] [4]. While model parallelism can work well in practice, data parallelism is the preferred approach for distributed DNN training. It has been the focus of more research, because it can be easily implemented and has good performance record. Data parallelism uses multiple replicas of a model, and distributes the training data across multiple

workers. Thereafter DL workers calculate gradients and send them to PS for model update [4].
There are two different approaches for parameter updating of distributed DNN training: Synchronous and Asynchronous. With synchronous method, a parameter server aggregates gradients of all workers then updates the model parameters at the end of training iteration. With asynchronous method, the parameter server immediately updates the model parameters whenever gradient arrives from a worker, without waiting to aggregate them collectively. The asynchronous method is advantageous because it can be performed faster without sacrificing accuracy compared to the synchronous method [1] [5]. This paper focuses on the study of parallelization techniques based on asynchronous fashioned data parallelism approaches.

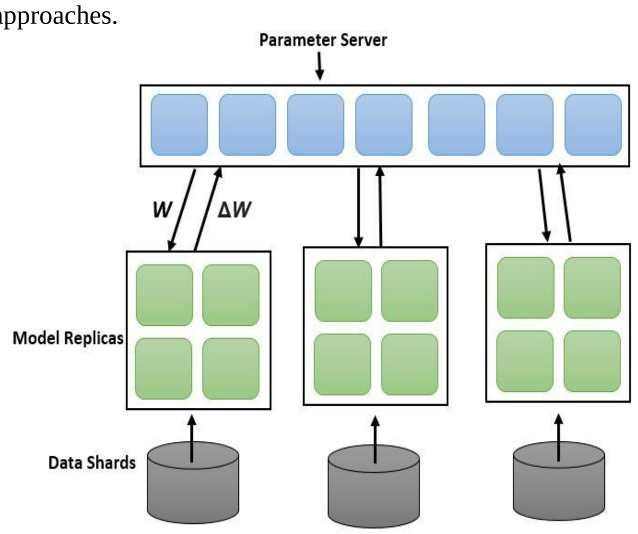

Figure - 1.1 Asynchronous distributed stochastic gradient descent.

Asynchronous distributed stochastic gradient descent model with some model replicas and common parameter server are explained in Figure-1.1. Data shared between the model replicas and updates, relayed to parameter server throughout the process asynchronously.

## II. RELATED WORK

Because of high memory footprint, to achieve better accuracy during training, considering very large-scale DNN model might be impossible to handle by a single computing node. Many ongoing research work are exploring distributed processing platforms such as High Performance Cluster for DNN training [13].
In Deep Learning model training Asynchronous SGD (ASGD) is one of the most widely used asynchronous distributed variants of Stochastic Gradient Decent algorithm. The ASGD has been proposed to address the disadvantage of Synchronous SGD where, workers have to wait until the slowest worker finishes calculating gradient [11, 12]. However, result of ASGD does not always guarantee a linear speedup of performance with increasing number of workers. Asynchronous SGD is limited in improving the actual training performance due to the delayed gradient problem. ASGD uses a parameter server to share parameters asynchronously in between workers. The Elastic Averaging Stochastic Gradient Decent (EASGD) method was proposed to maximize the benefits of DNN exploration. It allows each deep learning worker to maintain its own model replica as in any other Asynchronous SGD methods. The distributed training is performed by updating the global model with the moving average dynamically. In this method, unlike the Asynchronous SGD, the parameter server and the workers exchange the weight parameter learned by them and calculate the difference between the two weights vector concurrently, hence updating their own weight parameters by adding the scaled difference to it [2]. So far this method performs better than the Downpour SGD by decreasing the delay time of global weight updating between the parameter server and local working nodes.
Among the distributed deep learning optimization schemes based on the stochastic gradient descent (SGD), the Hogwild [16] showed that deep learning processes with same weight and different data set can train Deep Neural Network through asynchronous SGD model on shared memory architecture without locking. In addition, Dogwild framework an extension of the Hogwild framework, performs Deep Neural Network training by exchanging weights and gradients asynchronously between the master and worker node processes. The master process updates the global weights whenever it receives gradients from the slave processes concurrently, and shares the updated weights with all the slave processes at once. Each slave process updates the most recently received weights with the gradients which is explored by itself, and sends the gradients to the master [2][12]. Every time in each iteration worker node updates its parameter with master node or parameter server.

### *2.1 Parallel Programming:*

In this section we explore the basics of parallel programming techniques. To implement neural network in distributed fashion, understanding background of parallel programming is a pre-requisite. Parallel Programming is a technique to implement parallel algorithms on computers depending on the target architecture to exploit highest compute capability of the architecture available. They may range from simple threaded implementations to OpenMP on single machines. Accelerators or hardware like FPGAs are usually programmed with special languages such as CUDA, OpenCL etc. Yet, the details are often hidden behind library that implement the optimized programming primitives on these platforms. On machines with multiple node and distributed memory, one can either use simple communication mechanisms such as TCP/IP or Remote Direct Memory Access (RDMA) for internode communication. One can also use more convenient libraries such as the Message Passing Interface (MPI) [14].

### *2.2 Parallel Computing:*

In general parallel computing is a wide concept and refers to the process of computation where multiple calculations are carried out simultaneously with help of certain tools, technologies and hardware. In parallel computing large number of problems can often be divided into smaller sub problems which can then be solved simultaneously with help of independent compute component. There are several different types of parallel computing techniques such as instruction-level, bit-level, data, and task parallelism. Parallelism has long been employed in high performance computing (HPC), but it's gaining huge interest due to the physical constraints of building more advance processor due associated limitation of frequency scaling and power dissipation. Power consumption by computer system is a major concern; therefore parallel computing has become the dominant paradigm in computer architecture, mainly in the form of multi-core processors and new generation's commodity hardware components [19].

*2.3 DistBelief Model:*

The DistBelief framework proposed Downpour SGD, which supports a large number of model replicas, as a technique of asynchronous SGD. Within DistBelief, the developers proposed Sandblaster batch optimization method [3], supporting various distributed models. Each DNN training worker in an asynchronous SGD can learn weights at different speeds without synchronization overhead therefore maximize the utilization of computing and network resources. Specifically, the resources of a heterogeneous HPC system consisting of CPUs and GPUs of heterogeneous specifications (clock rate, the number of cores, etc.) can be utilized effectively to take advantage of the maximum computational performance [4].

So far the Distributed deep learning platforms such as Petuum [19] and Microsoft Cognitive Toolkit (CNTK) [18] uses distributed key-value repositories developed specifically for parameter servers [4][10]. Under this environment, parameter server manages asynchronous parameter updates among deep learning workers and provides the advantage of supporting an elastic coherence model, flexible scalability and fault tolerance. Virtual shared memory framework can accelerate the parameter sharing in distributed deep learning platform therefore can supports asynchronous SGD through experiments [7].

*2.4 SMB (Soft Memory Box):*

Soft Memory Box enables sharing the memory of remote node among distributed processes on the computing
nodes so as to improve communication performance via parameter sharing. The SMB consists of an SMB Server and SMB client components: SMB Library, SMB Device Driver, and Infiniband Communication Layer module. The SMB Server run on memory node and provides shared memory to distributed processes. A SMB client component runs on computing node. SMB Library, which is a user-level static library, provides application programming interfaces for distributed application processes and statically linked with the executable during compile time. SMB Device Driver and Infiniband Communication Layer (ICL) module are kernel-level modules and is loaded before the execution of application processes. The SMB Server make available a portion of physical memory of the memory node as shared memory buffer over the high speed RDMA enabled network by pooling it in advance, as well as on-demand by the client process [1,4,5].

*2.5 Caffe:*

Caffe is designed by Berkeley AI Research (BAIR) and the Berkeley Vision and Learning Center (BVLC) at UC Berkeley to provide expressive architecture and GPU support for deep learning applications [11]. It is based on C++, CUDA and can be operated through command line and Python. It runs on CUDA based devices, mobile platforms and additionally been extended for use in the Apache Hadoop ecosystem with Spark. Caffe employs a mini-batch-based stochastic gradient for training optimization: mini-batch Standard Gradient Descent (SGD) can perform training on independent data samples and then update the network parameters at the end of every N training samples [15].

*2.6 SHMCAFFE:*

ShmCaffe is a distributed deep-learning platform that uses remote shared memory based on SMB for parameter sharing, by extending BVLC Caffe (version 1.0.0). ShmCaffe does not only implements ASGD, but a hybrid method that combines Inter-node Asynchronous SGD and Intra-node Synchronous SGD. ShmCaffe memory model runs on distributed deep learning clusters with multiple nodes and exchanges initialization messages between the distributed processes using Message Passing Interface(MPI). In order to exchange parameters between the distributed Inter-node workers, the asynchronous EASGD algorithm is modified to use remote shared memory buffer [2].

## III. LARGE SCALE SHARED MEMORY ON HPC PLATFORM

In this section we inspect various programming paradigms, libraries, technologies already available and that may become strong alternative of SMB, and henceforth can support distributed deep learning algorithms like EASGD on HPC Clusters.

PGAS:

Partitioned global address space (PGAS) languages offers a programming abstractions similar to shared memory model, but with the control over data layout that is critical to high performance and scalability. Most common PGAS based languages include Unified Parallel C, Titanium (a scientific computing dialect of Java) and Co-Array FORTRAN (CAF).
Under PGAS, compilers uses a source-to-source translation strategy that converts parallel programming constructs to C with calls to communication layer like GASNet or MPI [14].

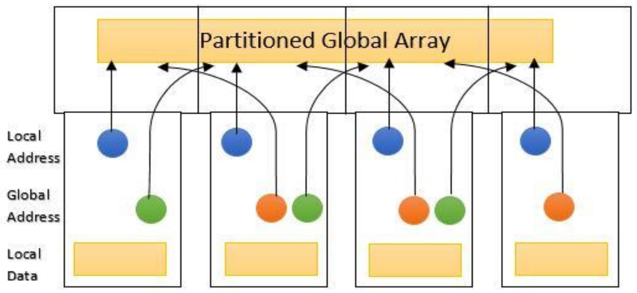

Fig. 3.1 PGAS Memory Model

*A. Programming Languages with PGAS*

*3.1 UPC:*

UPC is an extension of ISO C language that implements PGAS Programming model, defined by a consortium and supported by multiple proprietary and open source

compilers. Unified Parallel C exposes global shared address space to the computing instances which is logically divided among many threads and each thread is associated or shows affinity to a part of the shared memory. UPC also provides a private memory space per thread for local computations, as shown in PGAS memory model. Therefore each thread in the compute node has access to both its private memory and to the global shared space. In UPC, memory specification combines the advantages of both the shared and distributed memory programming models. Global shared memory space facilitates the development of parallel codes, where multiple parallel threads can access global shared memory space through high speed read/write protocol.
UPC is designed to perform best on large-scale parallel machines like HPC clusters. So far it combines the programmability advantages of the shared memory programming along with control over data layout without sacrificing performance aspect of MPI [13, 14].

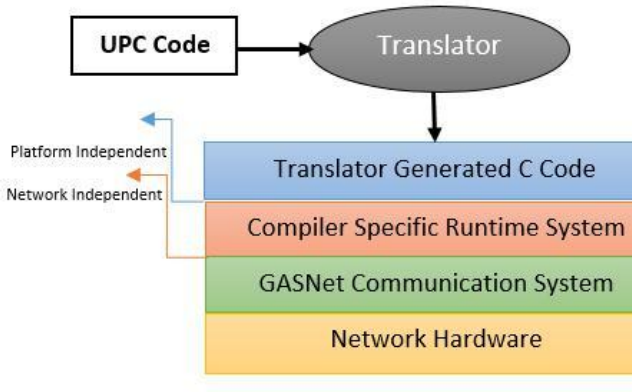
UPC Code
Translator
Platform Independent
Network Independent
Translator Generated C Code
Compiler Specific Runtime System
GASNet Communication System
Network Hardware

Fig. 3.2 : Unified Parallel C architecture

*3.2 UPC++:*
Unified Parallel C++ offer an easy on-ramp to PGAS programming through interoperability with other existing parallel programming systems. UPC++ comprise of following features such as Remote memory access (RMA), Remote Procedure Call (RPC), View based Serialization, Shared objects. It Expose useful asynchronous parallel programming expressions which are unavailable in traditional Single Program Multiple Data (SPMD) model such as remote function invocation, and continuation based operation completion for supporting complex scientific applications. Unified Parallel C++ follows object oriented programming model with PGAS memory model. Unified Parallel C++ covers so many features in parallel computing environment and it includes useful parallel programming key functions unavailable in UPC, for example this programming platform provides an asynchronous remote function invocation and multidimensional arrays also it is used to support complex scientific applications [20].

*GASNet:*
GASNet stands for "Global Address Space Networking". It is a language-independent networking middleware layer that provides, high-performance communication primitives including Remote Memory Access and Active Messages. It has been used to avail underlying networking mechanism for the programming models and libraries such as UPC, UPC++, Co-Array Fortran, Chapel, and Legion.

The design of GASNet middleware is partitioned into two layers to ease the porting experience without sacrificing performance. The lower layer is a narrow but very generic interface implementation, called GASNet core API. It is heavily based on Active Messages, and it is implemented directly on top of each individual network communication layer. The top level is a wider and more expressive interface which is called the GASNet extended API and provides high level operations such as remote memory access and various collective operations [14].

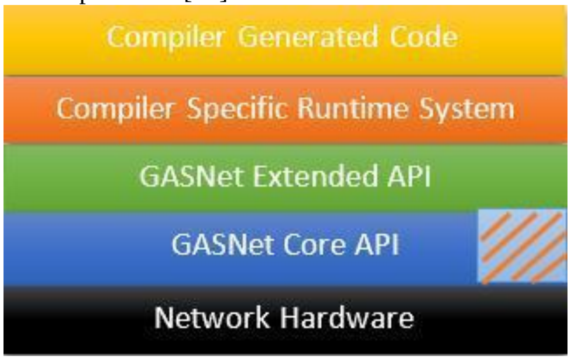

Fig. 3.3: GASNet Communication System

## IV. CONCLUSION

In this paper we have studied various distributed deep learning platform implementation on high performance computing system. This includes different virtual shared memory Framework, Libraries and Network Protocol. We also discussed parameter sharing approach with ASGD algorithm between master and worker nodes. Through this study, it is clear that to realize Distributed Deep Learning on HPC Platform, the De Facto programming framework like MPI may not suffice. To exchange frequent messages, which are a primary requirement of DDL, we require a software eco-system and programming framework like UPC. In this context PGAS programming model and associated implementation seems to have promising prospects. We also studied SMB Framework, which is a significant development towards DDL. In future we wish to do more detail study of PGAS-UPC and SMB to perform a comparative study of their prominent features. Networking middleware like GASNet is going to be another candidate of our future study. SMB Framework implements high performance network communication primitives over RDMA, whereas similar communication primitives like RMA & Active Messages (AM) has already been implemented in GASNet. A detail study into these aspects of SMB and GASNet would reveal more insight into how frequent parameter sharing can be achieved to make DDL possible. When computing is moving to exascale era, there is revived interest towards PGAS-UPC like global address space programming techniques. In our future work, we wish to see how the same techniques opens up new direction in Distributed Deep Learning implementation.

REFERENCES

[1] “Soft Memory Box: A Virtual Shared Memory Framework for Fast Deep Neural Network Training in Distributed High Performance Computing”, Shinyoung Ahn 1,2, Joongheon Kim 3, (Senior Member, Ieee), Eunji Lim2, And Sungwon Kang4, (Member, Ieee), 2018

[2] “ShmCaffe: A Distributed Deep Learning Platform with Shared Memory Buffer for HPC Architecture”, ShinyoungAhn, Joongheon Kim, 2018

[3] “Large Scale Distributed Deep Networks”, Greg S. Corrado, Jeffrey Dean, RajatMonga, Kai Chen, Andrew Y. Ng, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc AurelioRanzato, Andrew Senior, Paul Tucker, Ke Yang, Google

[4] “Deep learning with COTS HPC systems”, Adam Coates, David J. Wu, Andrew Y. Ng, Brody Huval, Tao Wang, NVIDIA Corporation.

[5] “Parallel I/O optimizations for scalable deep learning”, SarunyaPumma, Min Si, Wu-chun Feng, and PavanBalaji, 2017IEEE

[6] “iRDMA: Efficient Use of RDMA in Distributed Deep Learning”, Systems Yufei Ren∗, Xingbo Wu†, Li Zhang∗, Yandong Wang∗, Wei Zhang, 2017 IEEE

[7] “A- Distributed Deep Reinforcement Learning on the Cloud for Autonomous Driving”, Mitchell Spryn,Aditya Sharma, 2018 ACM

[8] “UPC++: A PGAS Extension for C++”, Yili Zheng, Amir Kamil, Michael B. Driscoll, Hongzhang Shan, Katherine Yelick,

[9] “Big Data and Deep Learning”, B.M. Wilamowski, Bo Wu, 2016 IEEE

[10] “Acelerating Training of DNN in Distributed Machine Learning system with Shared Memory”, Eun-jin Lim 2013, Shin Young Ahn, 2017IEEE.

[11] "BAIPAS: Distributed Deep Learning Platform with Data Locality and Shuffling", Mikyoung Lee, Sungho Shin, Seungkyun Hong, Sa-kwang Song, 2017, ECEECS.

[12] "Distributed Deep Learning Framework based on Shared Memory for Fast Deep Neural Network Training", Eun-Ji Lim, Shin-Young Ahn, Yoo-Mi Park, Wan Choi, 2018 IEEE.

[13] "Performance Modeling and Evaluation of Distributed Deep Learning Frameworks on GPUs", Shaohuai Shi, Qiang Wang, Xiaowen Chu, 2018 IEEE.

[14] "Productivity and Performance Using Partitioned Global Address Space Languages", Katherine Yelick, Dan Bonachea1, Wei-Yu Chen, Phillip Colella.

[15] “NUMA-Caffe: NUMA-Aware Deep Learning Neural Networks”, Probir Roy, Shuaiwen Leon Song, Sriram Krishnamoorthy, Abhinav Vishnu, Intel Labs XU LIU, College of William and Mary, 2018 ACM.

[16] “HogWild++: A New Mechanism for Decentralized Asynchronous Stochastic Gradient Descent” Huan Zhang, Cho-Jui Hsieh, Venkatesh Akella, 2018.

[17] “Re-designing CNTK Deep Learning Framework on Modern GPU Enabled Clusters”, Dip Sankar Banerjee, Khaled Hamidouche, and Dhabaleswar K. (DK) Panda, IEEE 2016.

[18] “Petuum: A New Platform for Distributed Machine Learning on Big Data”, Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng.

[19] “Evaluating and Modeling Power Consumption of Multi-Core Processors”, Robert Basmadjian, Hermann de Meer, 2012, ACM.