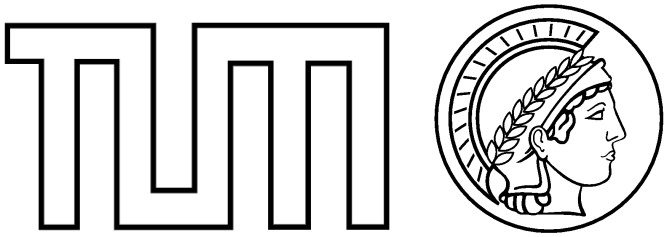

Max Planck Computing and Data Facility
Chair of Computer Architecture and Parallel Systems

Tensorflow

Seminar: Efficient Programming of HPC Systems
Frameworks and Algorithms

Efe Kamasoglu

May 30, 2023

Contents

Abstract

1 Introduction

TensorFlow is a free and open-source framework developed by Google Brain, which finds its application widely in the field of machine learning and artificial intelligence. It is used to build and train large-scale models according to the client's preferences and provided data sets. In order to train a model, TensorFlow carries out several computations by mapping them onto a variety of hardware, such as mobile devices or systems consisting of multiple computational units with hundreds of GPUs. Those computations are represented in a "directed dataflow graph"[1] as a whole, which is then compiled statically or dynamically depending on the version of Tensorflow [1]. A graph can be extremely large due to massive real-world training sets. That is why we have to utilize HPC clusters in the first place to efficiently process the data.

To construct such graph, a client needs to create a session which is an interface with methods of its own. Through those methods, the client can introduce his data set to Tensorflow's system, define his model and its specifications. A graph is typically composed of the following [1]:

- *Node:* A node has zero or more inputs and zero or more outputs. It is an instantiation of an operation.
- *Edge:* Data flows through the edges from one node to another. There are also edges which are not for the dataflow, but for control dependencies between different nodes; e.g., execution order of the concerning operations.
- *Operation:* An operation represents a computation such as addition or matrix dot product.
- *Tensor:* Tensors are multidimensional arrays describing the data. They flow into the nodes as inputs through the edges of a graph.
- *Kernel:* A kernel is an implementation of an operation.
- *Device:* Devices are computational units on a machine that are utilized by TensorFlow's system to execute kernels. (e.g. GPUs)

An example graph for a neural network is shown in Figure 1. W, x are tensors which flow into 1. node with the operation $MatMul$ as inputs. Matrix multiplication of W and x is then computed and the result as well as tensor b are fed into 2. node. Addition of those flow into 3. node, where the $ReLU$-function is applied on the addition. In the end, final node is executed, thus cost of the trained model C is computed and returned.

Client can either execute all or a part of the graph with the help of the session, where he interacts with the processes that regulate access to devices. In

[1] Martin Abadi, Ashish Agarwal, ..., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," November 9, 2015, p. 1, para. 2

addition to that, TensorFlow system itself has different implementations for execution mechanisms both for single- and multiple-device systems [1].

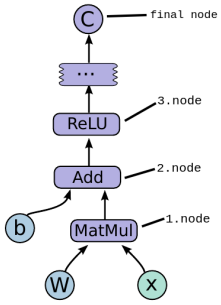

Figure 1: Subgraph example for a neural network[2]

2 Execution of a Graph

Each implementation follows the same standard (Fig. 2): Client process calls the master process through the session interface to ignite the execution. An arbitrary number of worker processes, each of which executes a subgraph, are called by the master process. Each worker is responsible for one or more devices. Master distributes the operations as well as the data tensors to the workers. Also, a worker can fetch the data by itself directly from storage system to the device's memory. The number of workers depends on the architecture of the system. As we are going to focus on efficiency, scalability and portability of TensorFlow in terms of HPC systems, we have to take a closer look at the distributed implementation [1].

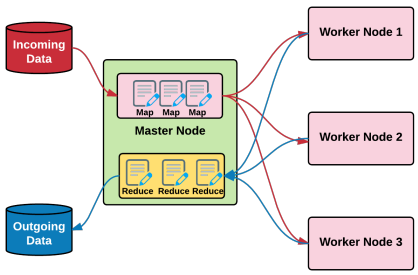

Figure 2: Master and Worker processes of a distributed system[3]

[2]Martin Abadi, Ashish Agarwal, ..., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," November 9, 2015, p. 3, fig. 2

[3]https://www.oreilly.com/api/v2/epubs/9781787281349/files/assets/Ch09-Fig11.png

3 Distributed (Multiple-Device) Implementation

If we consider a system with a single device, the execution is fairly simple: A worker process is created for the device. Data is placed into the memory and all the nodes are mapped to the same device. The device executes them in an order according to the control dependencies between the nodes. However, in the case of an HPC system with multiple devices, we encounter a handful of programming challenges as to resource management, mapping of the nodes to the devices and intercommunication between the devices in terms of data transfer.

3.1 Resource Management & Mapping of the Nodes

In large-scale machine learning, we have to deal with massive data sets over thousands of petabytes. In order to train a model on such data set, hence construct a dataflow graph, we need to parallelize our node execution and also divide our data into subsets across multiple devices of our system.

TensorFlow uses a greedy algorithm for mapping the nodes: Traverse the graph according to the dependencies between the nodes. At each node, the execution is simulated

3.2 Intercommunication between Devices

References

[1] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg,...Xiaoqiang Zheng (November 9, 2015): TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems

[2] Leslie Lamport (1994) *LaTeX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.