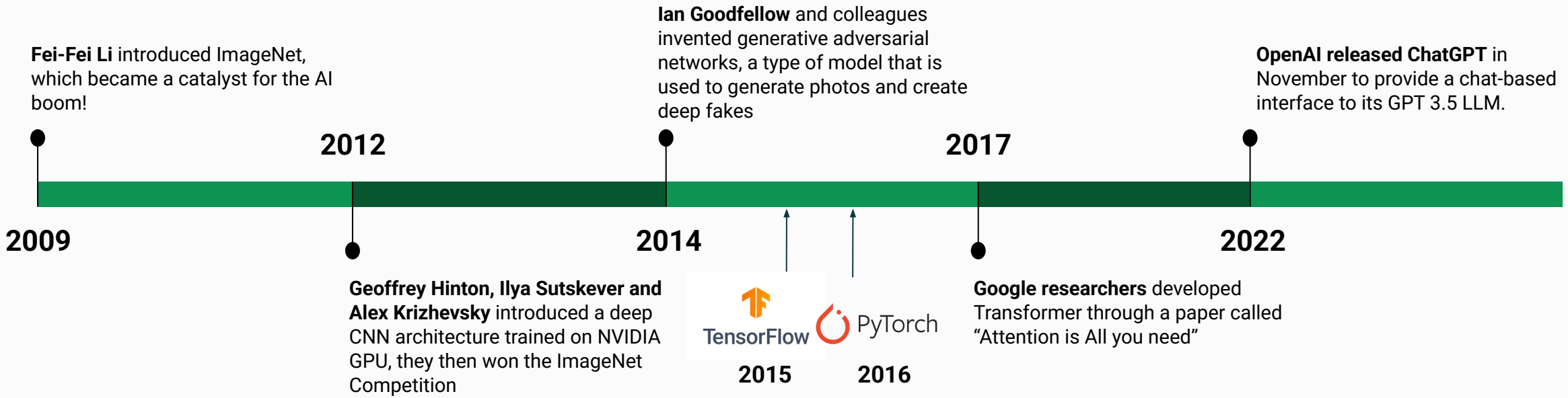
UNIVERSITY OF CAMBRIDGE

Topic: M. Abadi et al.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Preliminary White Paper, 2015.

Presenter: Woon

UNIVERSITY OF CAMBRIDGE

What does Machine Learning actually needs? (Pre-2015 Challenge)



1. Flexibility to express diverse ML algorithms
2. Heterogeneous Hardware Support
3. Scalability
4. Research-to-Production Pipeline

UNIVERSITY OF CAMBRIDGE

The Dataflow Programming Model - Computation as a Graph

Dataflow programming

10 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

In computer programming, **dataflow programming** is a programming paradigm that models a program as a directed graph of the data flowing between operations, thus implementing dataflow principles and architecture.[1]

Core Principles of Dataflow Programming:

1. Computation == Directed Graph
2. Data-Driven Execution
3. Explicit Data Dependencies
4. Implicit Parallelism

2 Stage Process in TensorFlow:

1. Define program as dataflow graph
2. Optimize then execute

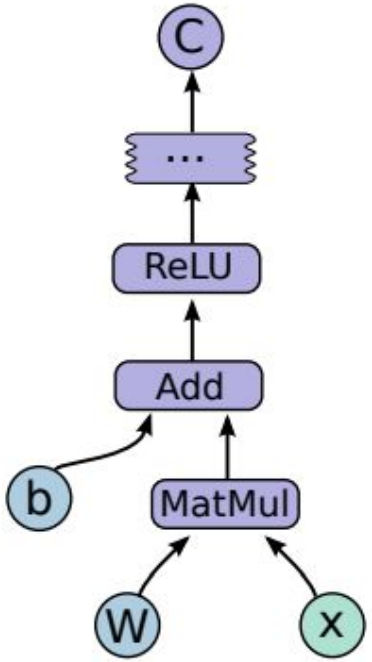


Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

General-Purpose Dataflow Wasn't Enough

Dimension	Earlier Dataflow System (Pre-Tensorflow)	What Machine Learning in DataFlow needs?
Primary Use Case	• Focus on Batch Data Processing	• Iterative Optimization on Model Training
Execution Pattern	• One-shot jobs or periodic resubmission	• Millions of iterations in single session
State Management	• Mostly external storage	• Persistent mutable parameters in graph
Graph Structure	• Acyclic (DAGs only)	• Cyclic (loops) + conditionals
Operation Granularity	• Coarse (map/reduce on partitions)	• Fine-grained (individual matrix ops)

UNIVERSITY OF CAMBRIDGE

Lessons from TensorFlow's Predecessor - DistBelieve

Before Tensorflow, Google was using DistBelieve for their research.

Limitation #1: Separate Training and Inference Systems
Limitation #2: Hard-Coded Neural Network Focus, Difficult to express other ML
Limitation #3: Parameter Server as Separate Subsystem
Limitation #4: Limited Flexibility in Parallelism
Limitation #5: Difficult Debugging and Visualization

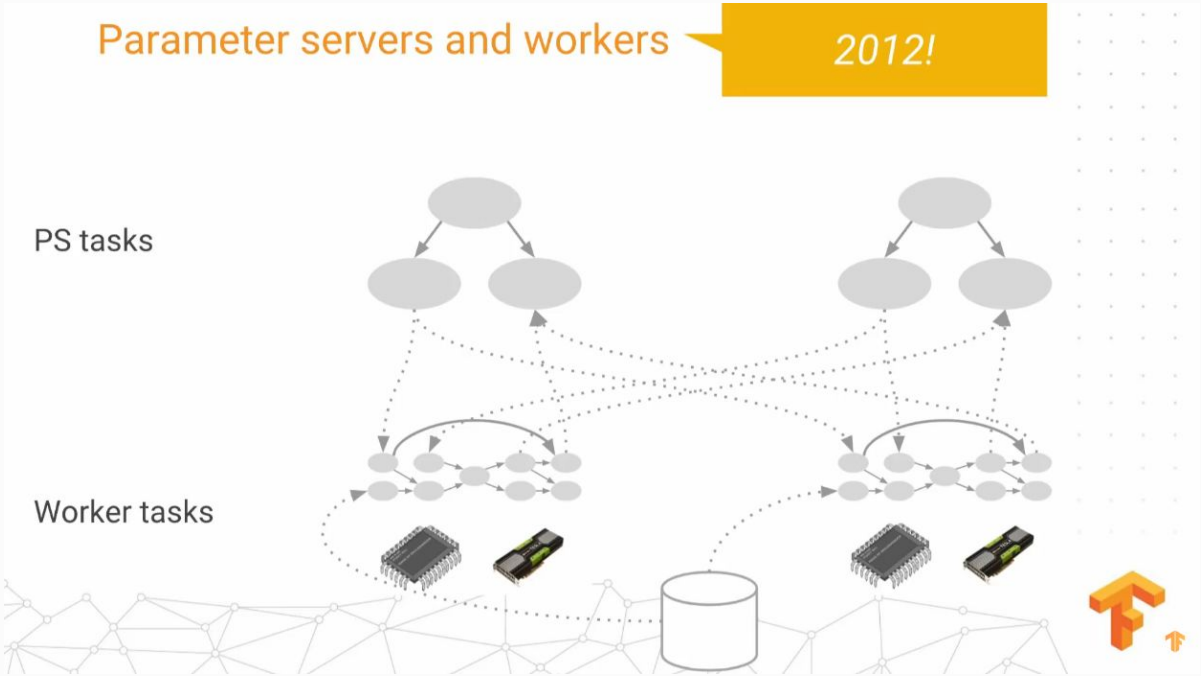


Image Source: Distributed TensorFlow (TensorFlow Dev Summit 2018)
https://www.youtube.com/watch?v=-h0cWBiQ8s8

UNIVERSITY OF CAMBRIDGE

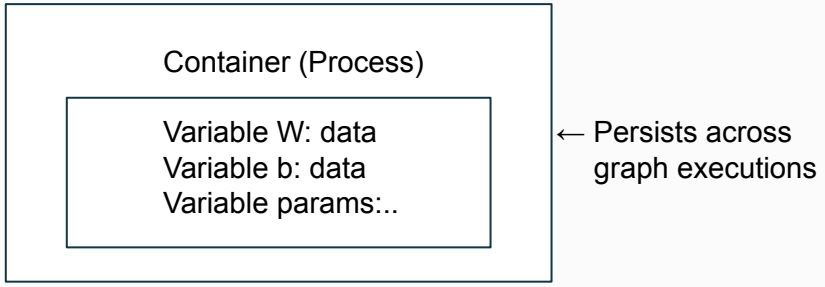
TensorFlow's Core Innovation #1 - Stateful Dataflow for ML

Variables

In most computations a graph is executed multiple times. Most tensors do not survive past a single execution of the graph. However, a *Variable* is a special kind of operation that returns a handle to a persistent mutable tensor that survives across executions of a graph. Handles to these persistent mutable tensors can be passed to a handful of special operations, such as *Assign* and *AssignAdd* (equivalent to +=) that mutate the referenced tensor. For machine learning applications of TensorFlow, the parameters of the model are typically stored in tensors held in variables, and are updated as part of the *Run* of the training graph for the model.

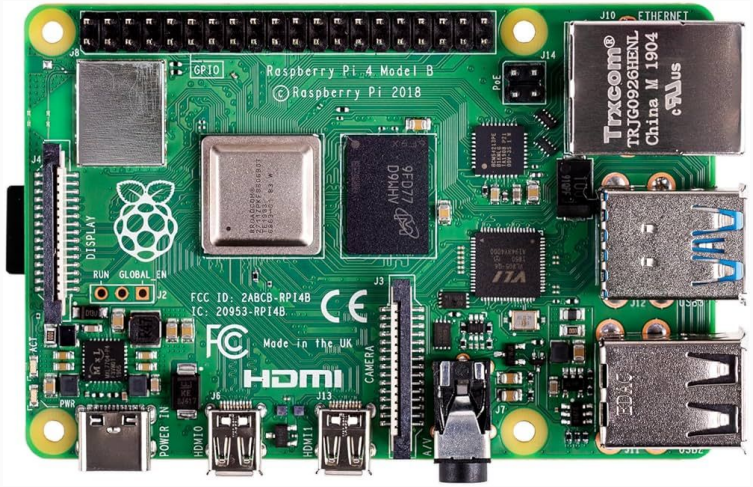
4.7 Containers

A *Container* is the mechanism within TensorFlow for managing longer-lived mutable state. The backing store for a *Variable* lives in a container. The default container is one that persists until the process terminates, but we also allow other named containers. A container can be reset by clearing it of its contents entirely. Using containers, it is possible to share state even across completely disjoint computation graphs associated with different Sessions.



UNIVERSITY OF CAMBRIDGE

TensorFlow's Core Innovation #2 - Single System, Any Platform



Operations and Kernels

An *operation* has a name and represents an abstract computation (e.g., "matrix multiply", or "add"). An operation can have *attributes*, and all attributes must be provided or inferred at graph-construction time in order to instantiate a node to perform the operation. One common use of attributes is to make operations polymorphic over different tensor element types (e.g., add of two tensors of type float versus add of two tensors of type int32). A *kernel* is a particular implementation of an operation that can be run on a particular type of device (e.g., CPU or GPU). A TensorFlow binary defines the sets of operations and kernels available via a registration mechanism, and this set can be extended by linking in additional operation and/or kernel definitions/registrations. Table 1 shows some of the kinds of operations built into the core TensorFlow library.

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

TensorFlow Advanced Feature #1: Automatic Differentiation

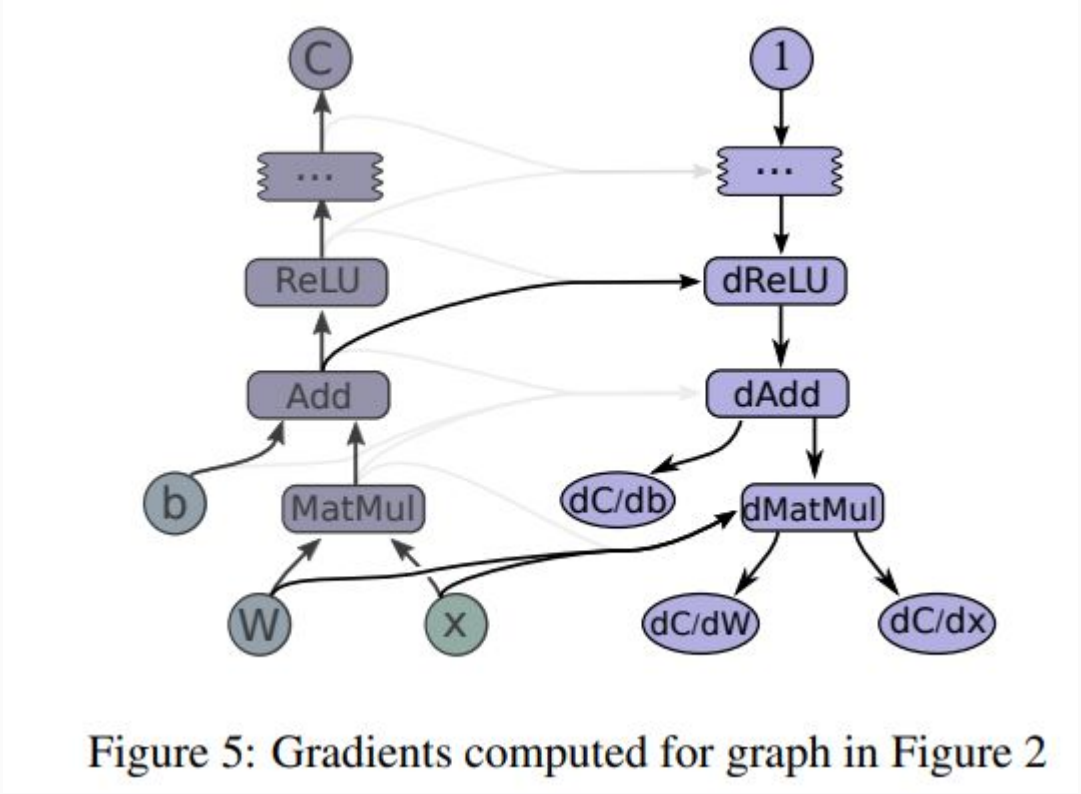


Figure 5: Gradients computed for graph in Figure 2

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

TensorFlow Advanced Feature #2: Partial Execution

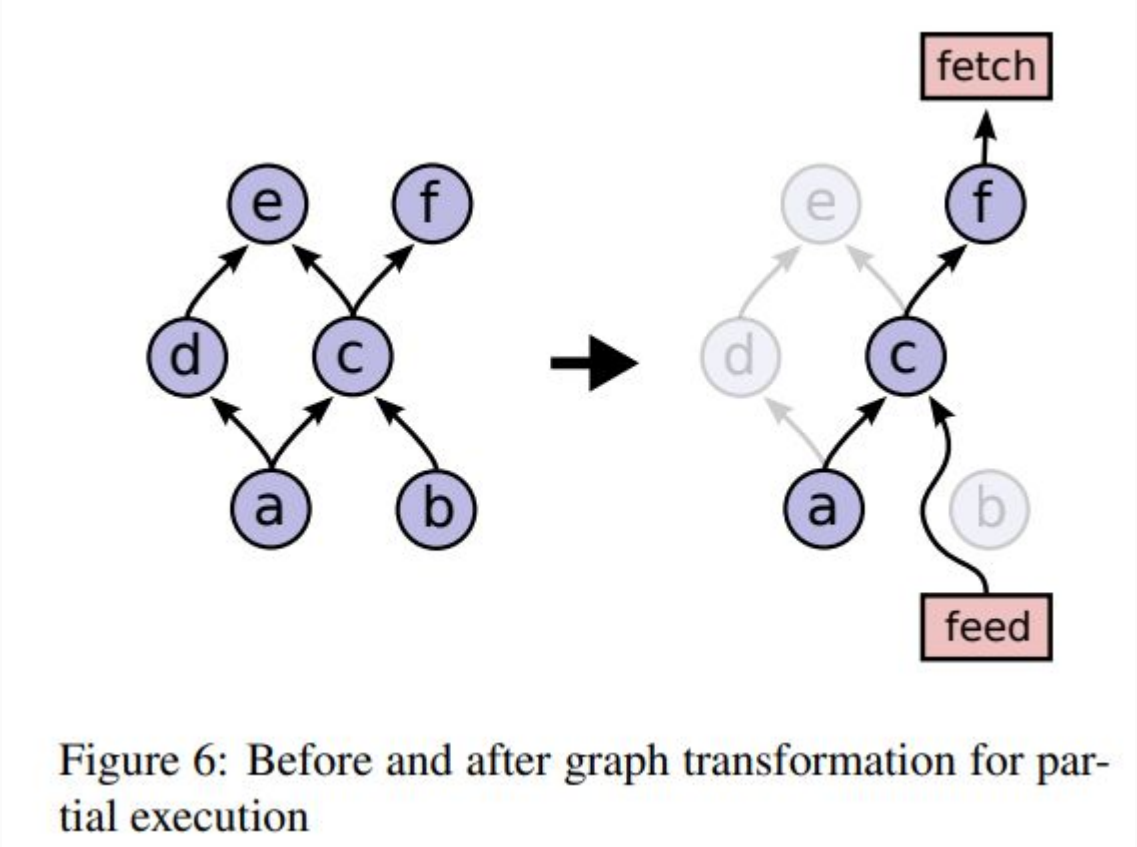


Figure 6: Before and after graph transformation for partial execution

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

TensorFlow Advanced Feature #3: Control Flow

Control flow operations	Merge, Switch, Enter, Leave, NextIteration

"We introduce a small set of primitive control flow operators... The Switch and Merge operators allow us to skip the execution of an entire subgraph based on the value of a boolean tensor. The Enter, Leave, and NextIteration operators allow us to express iteration"

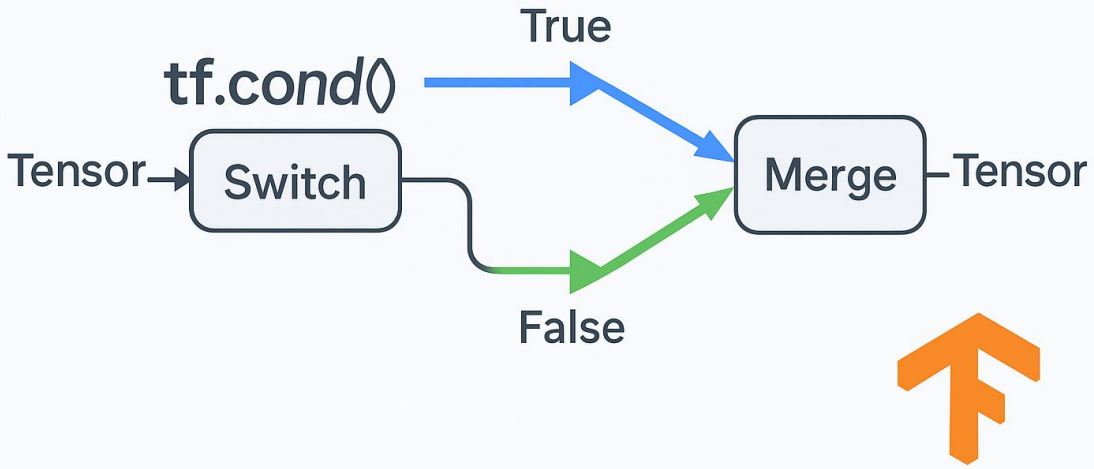


Image Source:
https://hfooladi.github.io/posts/2019/05/TensorFlow-Condition/

UNIVERSITY OF CAMBRIDGE

How Tensorflow scales to Hundreds/Thousands of Machines?

Challenge #1: Decide which device to place the computation for each node in the graph
Challenge #2: Manage the required communication of data across device boundaries implied by these placement decisions
Challenge #3: Handle failures without losing training progress

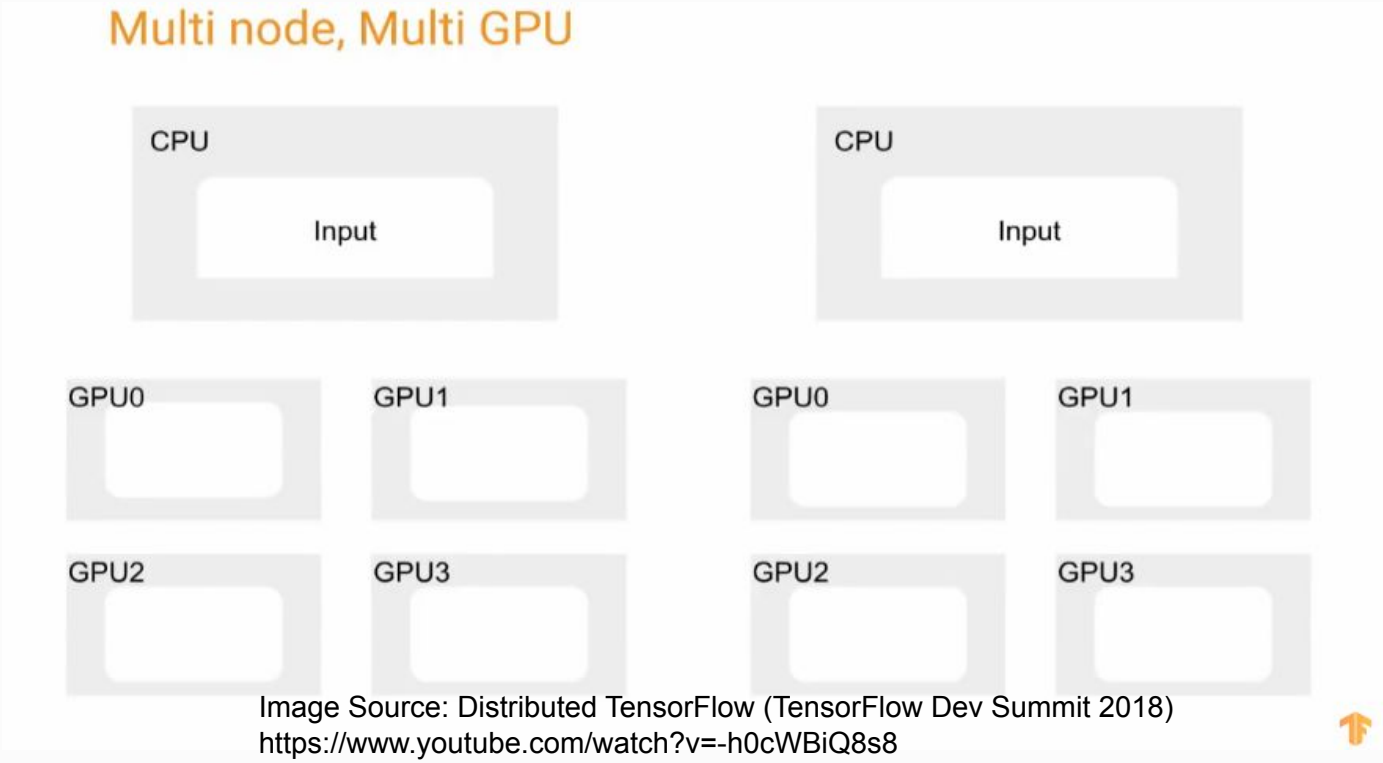


Image Source: Distributed TensorFlow (TensorFlow Dev Summit 2018)
https://www.youtube.com/watch?v=-h0cWBiQ8s8

UNIVERSITY OF CAMBRIDGE

Cost Model as input to a placement algorithm

Challenge #1: Decide which device to place the computation for each node in the graph

Step 1: Build a cost model which contains the estimated size of the input / output tensor & computational time.
Step 2: The placement algorithm runs a simulated execution of the graph

- Algorithm starts with the sources of the computation graph, and simulates the activity on each device in the system as it progresses
- Consider a set of feasible devices with suitable kernels
- For multiple feasible devices, the algorithm will pick device for each node in the graph using greedy heuristics
- This heuristic takes into account the estimated or measured execution time of the operation on that kind of device from the cost model

“On-going development”

UNIVERSITY OF CAMBRIDGE

From Parameter Server to Send / Receive Nodes (2015)

Challenge #2: Manage the required communication of data across device boundaries implied by these placement decisions

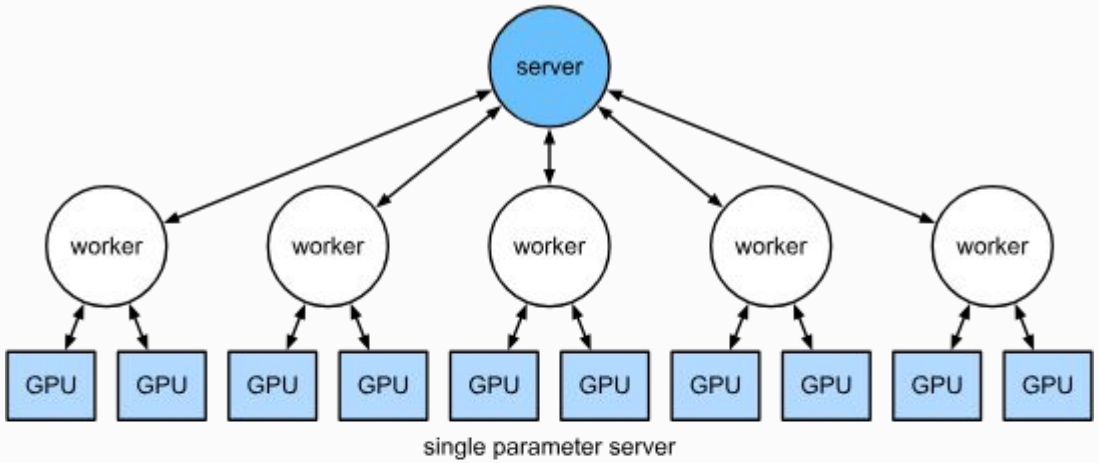


Image Source: https://d2l.ai/chapter_computational-performance/parameterserver.html

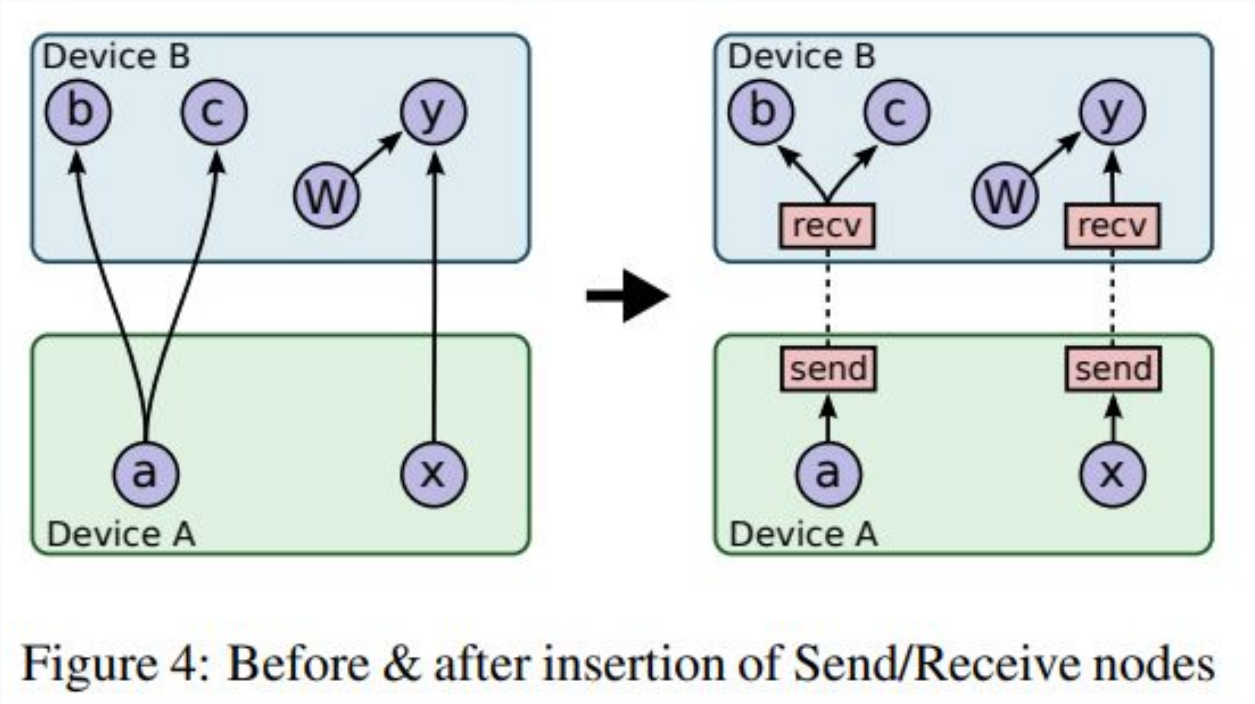


Figure 4: Before & after insertion of Send/Receive nodes

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

From Send / Receive Nodes to All Reduce (2018)

Challenge #2: Manage the required communication of data across device boundaries implied by these placement decisions

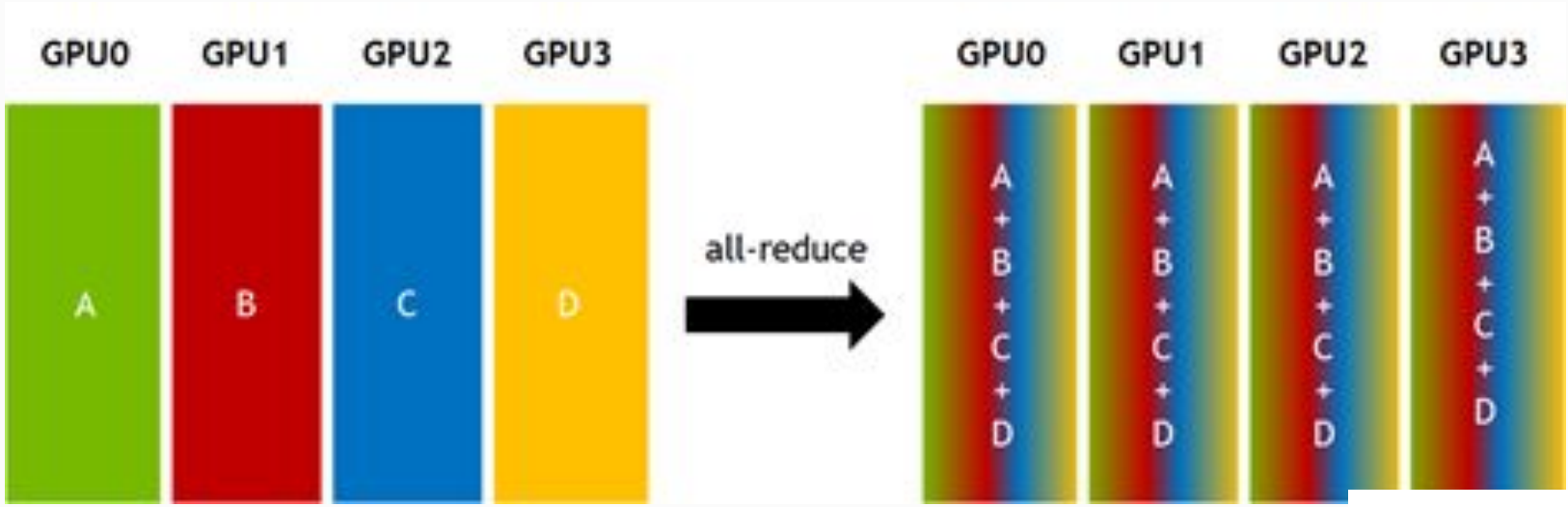


Image Source: https://developer.nvidia.com/blog/fast-multi-gpu-collectives-nccl/

All reduce is an collective operation, where all node contributes data, and every node receives the same aggregated data.

In machine learning, all-reduce operations are commonly seen in synchronous parameter updates of the distributed Stochastic Gradient Descent (SGD) optimization.

All reduce operation is included in NCCL, workable on NVLink and Infiniband.

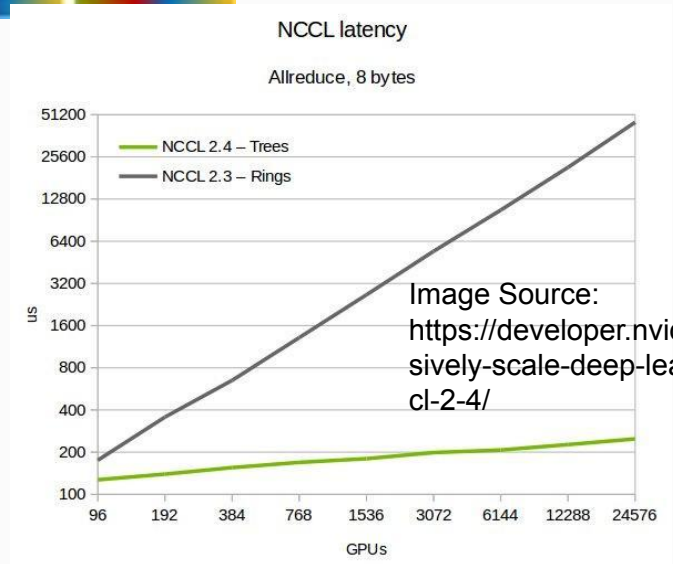


Image Source: https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/

UNIVERSITY OF CAMBRIDGE

Fault Tolerance & Checkpointing so Training Doesn't Stop for Failures

Challenge #3: Handle failures without losing training progress

Fault Tolerance

Failures in a distributed execution can be detected in a variety of places. The main ones we rely on are (a) an error in a communication between a Send and Receive node pair, and (b) periodic health-checks from the master process to every worker process.

When a failure is detected, the entire graph execution is aborted and restarted from scratch. Recall however that Variable nodes refer to tensors that persist across executions of the graph. We support consistent checkpointing and recovery of this state on a restart. In partcular, each Variable node is connected to a Save node. These Save nodes are executed periodically, say once every N iterations, or once every N seconds. When they execute, the contents of the variables are written to persistent storage, e.g., a distributed file system. Similarly each Variable is connected to a Restore node that is only enabled in the first iteration after a restart. See Section 4.2 for details on how some nodes can only be enabled on some executions of the graph.

UNIVERSITY OF CAMBRIDGE

Data Parallel Training

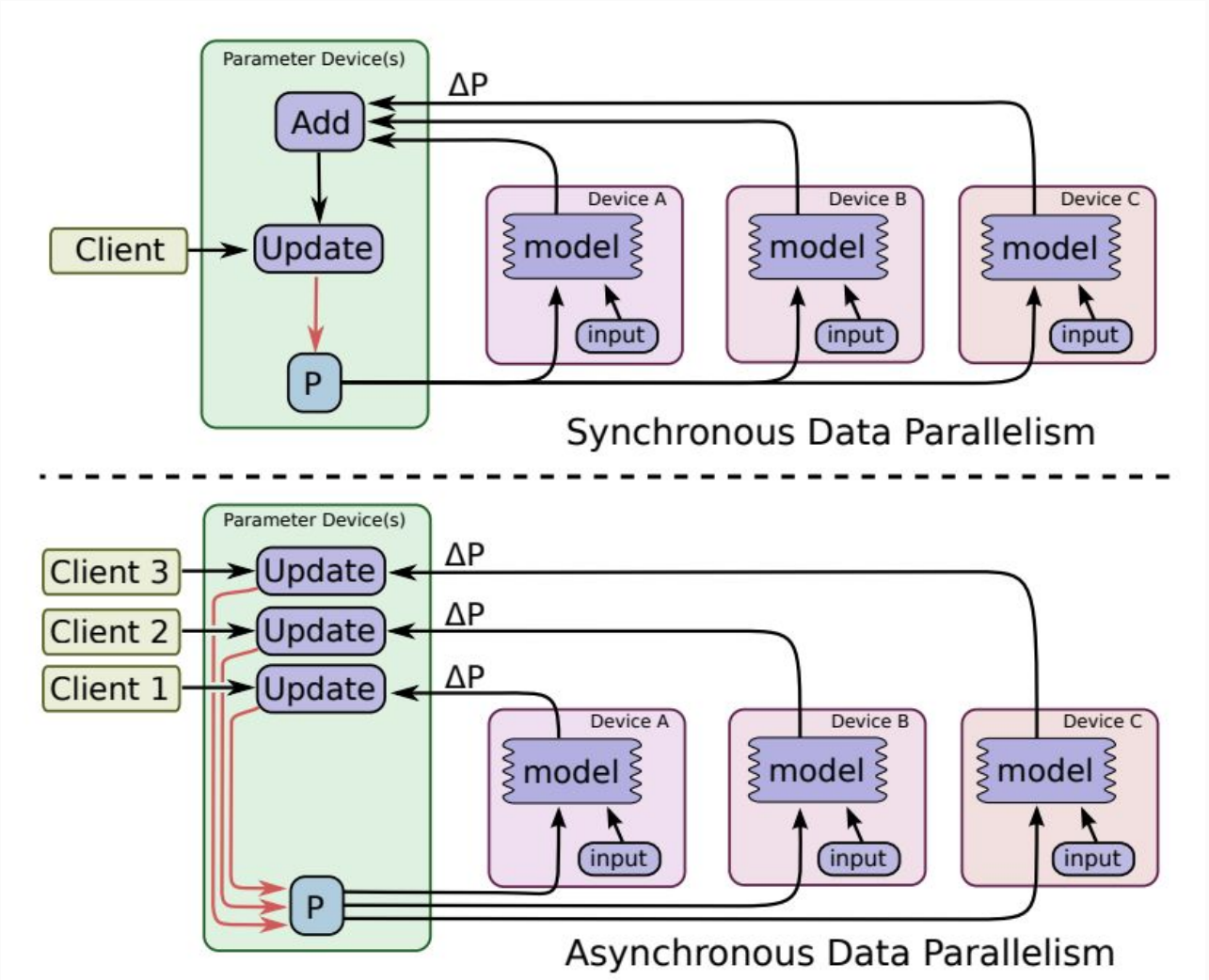


Figure 7: Synchronous and asynchronous data parallel training

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

Model Parallel Training

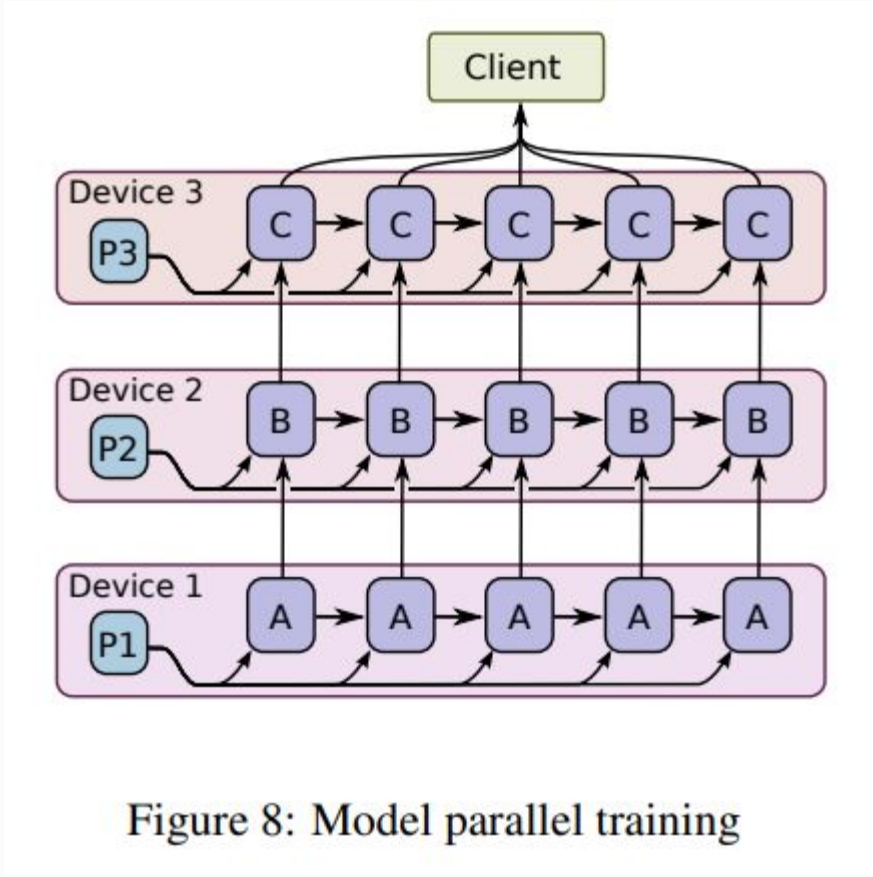


Figure 8: Model parallel training

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

Performance Optimizations

- Common subexpression elimination: Deduplicate identical ops
- Memory scheduling: reduce the time window during which intermediate results need to be kept in memory
- Kernel libraries: cuDNN, Eigen (don't reinvent optimized linear algebra)
- Lossy compression: 32-bit → 16-bit for cross-device transfers

UNIVERSITY OF CAMBRIDGE

Profiling & Visualization Tools

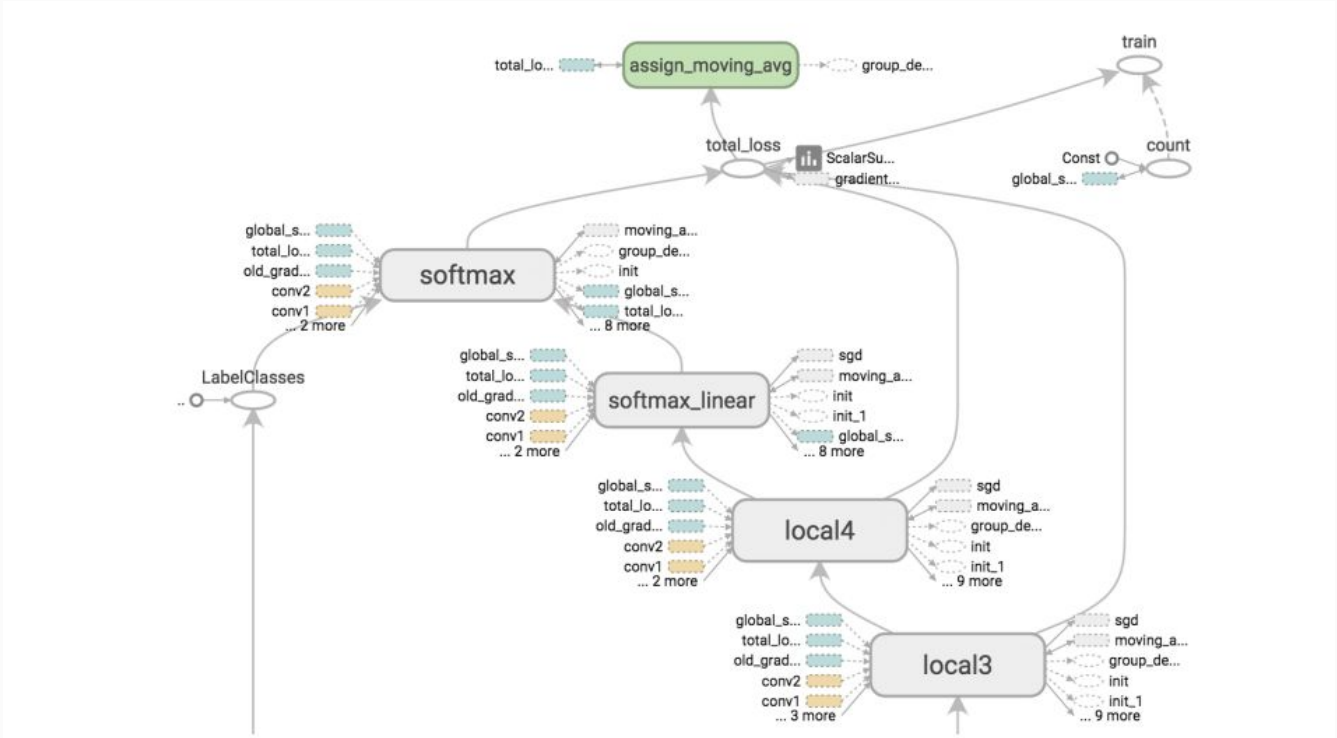


Figure 10: TensorBoard graph visualization of a convolutional neural network model

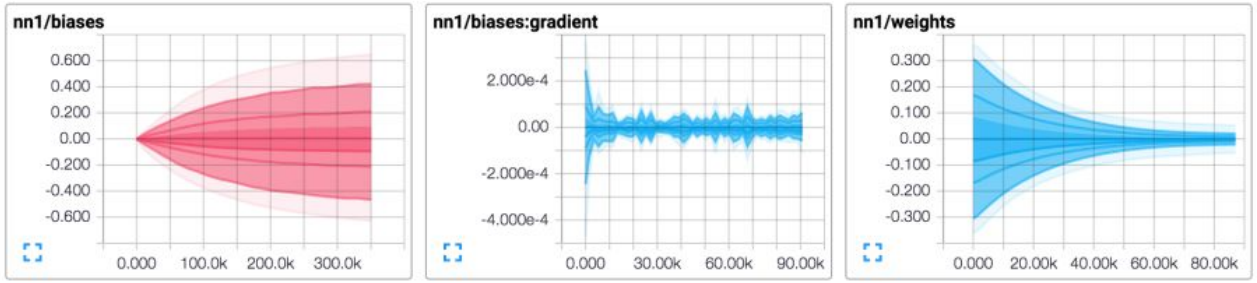


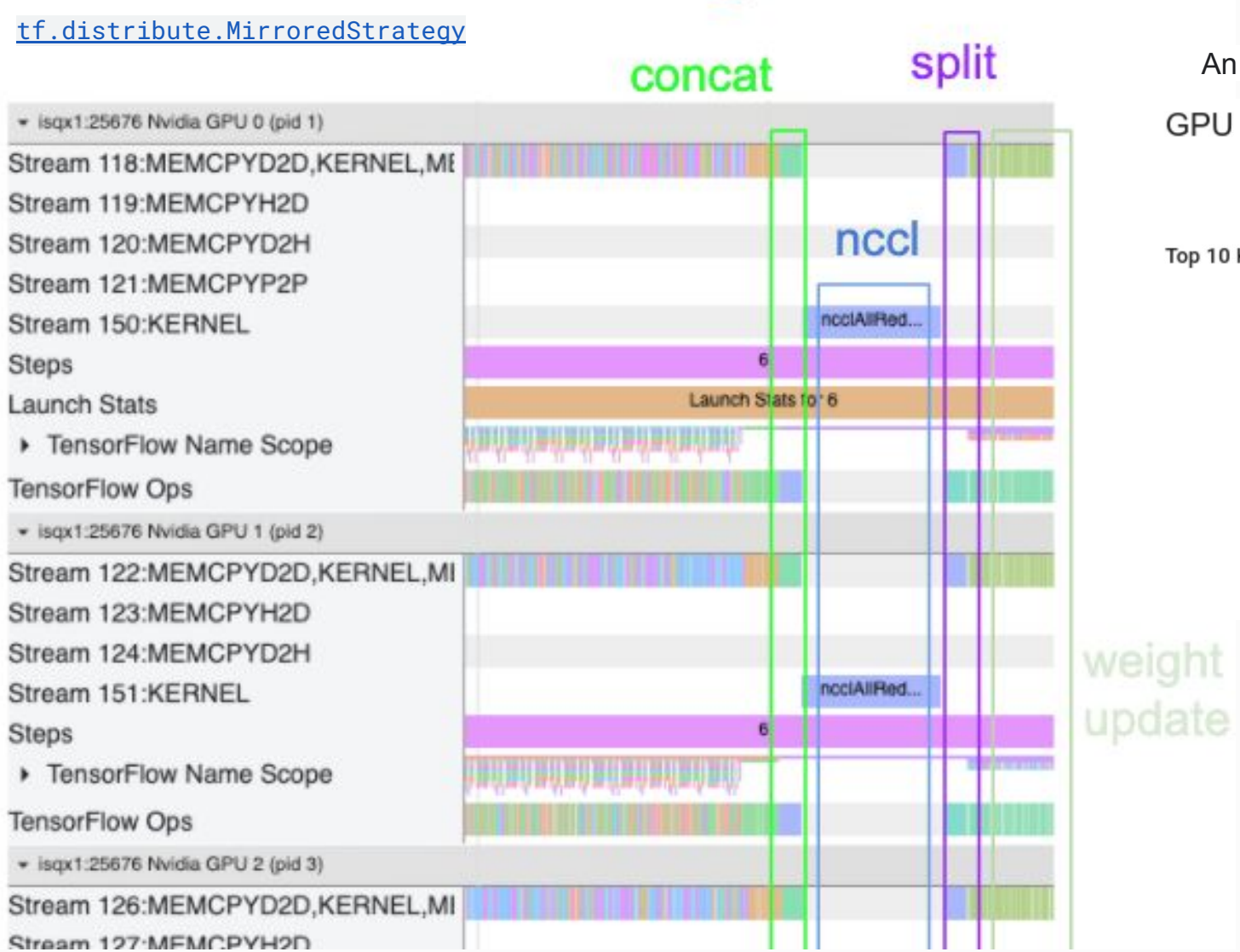
Figure 11: TensorBoard graphical display of model summary statistics time series data

Image Source: (Abadi et al., 2015)

UNIVERSITY OF CAMBRIDGE

Real-World Results

Optimize the performance on the multi-GPU single host using [tf.distribute.MirroredStrategy]



An example of communication bottleneck:

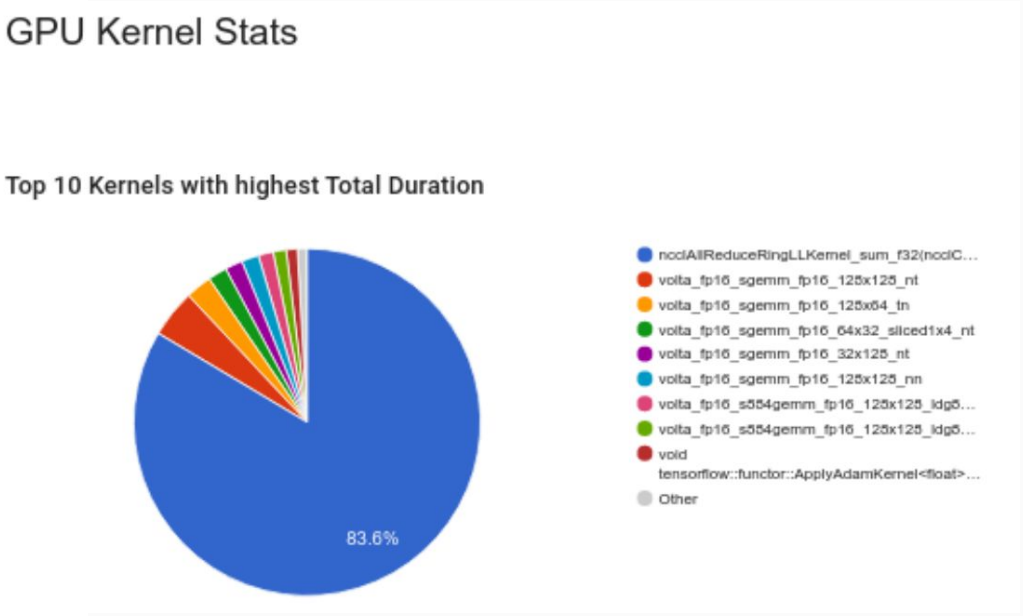


Image Source: https://www.tensorflow.org/guide/profiler

UNIVERSITY OF CAMBRIDGE

Modern Distributed Training, and How far we've come

- 2015: 100 machines, days for ImageNet
- 2025: 10,000+ GPUs, models with 100B+ parameters
- Technologies: AllReduce, mixed precision, gradient accumulation
- TensorFlow → PyTorch dominance (Eager Execution is easier to debug)
- Profiling shows: Communication still the bottleneck at scale

UNIVERSITY OF CAMBRIDGE

Thank you

Questions?