# aiXcelerate 2024

Using TensorFlow and Horovod

NHR4CES NHR for Computational Engineering Science | HPC.NRW | EDIH Rheinland | WEST AI KI-Servicezentrum | IT Center | RWTH AACHEN UNIVERSITY

# Agenda

- **TensorFlow Overview**
- **Distributed Deep Learning with TensorFlow**
  - Parallelism
  - `MirroredStrategy`
  - `MultiWorkerMirroredStrategy`
  - Horovod
- **How to use it on CLAIX**
  - Containers
  - Virtual environments

# TensorFlow – Overview

- **TensorFlow and Keras**
  - TensorFlow usually more low level. Keras used as higher-level abstraction
  - Sequential API and Functional API
  - Predefined models and datasets
  - Extensive documentation, tutorials and guides

- **Pros**
  - No manual training loop
  - Large community
  - Speed?

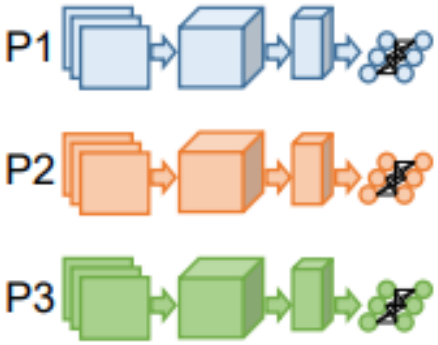
- **Cons**
  - A lot of changes to interfaces (limited compatibility between versions)
  - Native distributed solution hacky
  - Sometimes tricky to install in virtual environments

TensorFlow Keras

```python
import tensorflow as tf

# load and preprocess dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), _ = mnist.load_data()
ds_train = tf.data.Dataset.from_tensor_slices(x_train, y_train)

# create and compile model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')])

model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

# train the model
model.fit(ds_train, epochs=5)
```

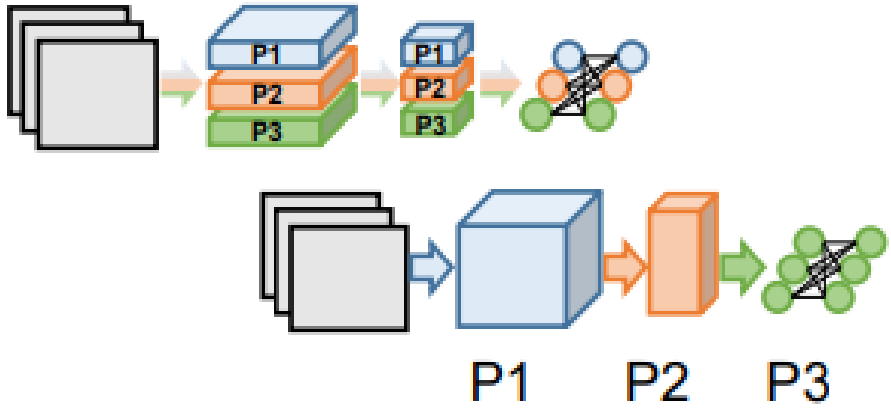
# Distributed Deep Learning – Strategies for Parallelism

- **Three most common strategies**




- **Data Parallelism**
  - Split data in micro-batches
  - Easy to implement
  - Best strong scalability

**Faster training**



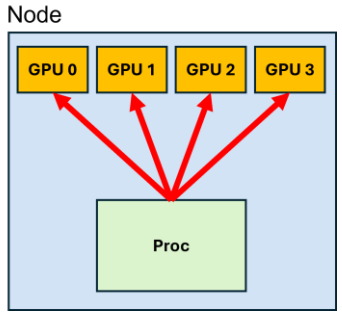

- **Model Parallelism**
  - Split layers in partitions
  - Necessary for large layers
  - Can improve load balancing
  - Hard and time consuming to implement

**Bigger models**

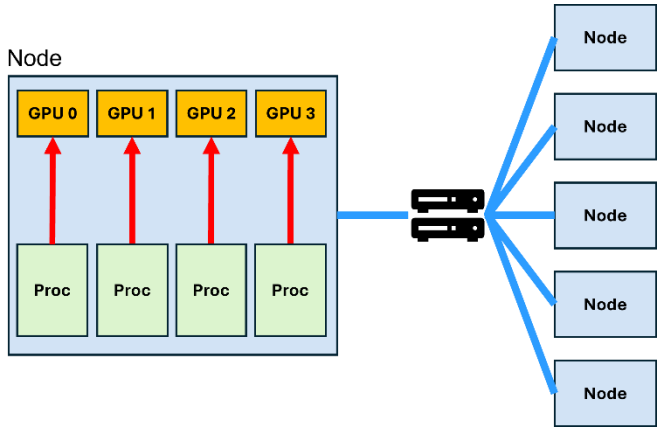
Images: Ben-Nun et al.

# TensorFlow – Distributed Deep Learning

- **Several ways to run distributed** (Documentation)
  - **Case:** Single node with multiple GPUs
    - Use `tf.distribute.MirroredStrategy` (Documentation)
    - Single process that is serving GPUs
    - Each GPU gets own copy of the model / network

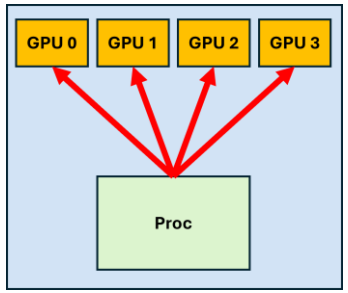



  - **Case:** Utilize multiple GPUs on one or more nodes
    - Use `tf.distribute.MultiWorkerMirrored` (Documentation)
    - Separate processes that each serve 1 GPU
    - Each GPU gets own copy of the model / network
    - Setup tricky
    - Use Horovod on top of TensorFlow (Documentation)
    - Wraps optimizer for backward propagation → `DistributedOptimizer`



# TensorFlow – Distributed Examples (1)

- **MirroredStrategy**

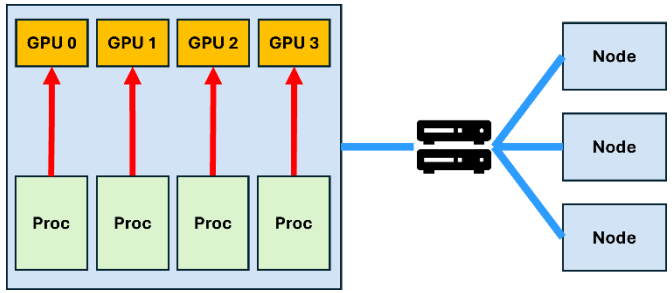



  - Further steps required
    - Define distributed strategy
    - Create and compile model under strategy scope
  - TensorFlow will automatically take care of
    - Data sharding
    - Weight exchanges

```python
import tensorflow as tf

# load and preprocess dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), _ = mnist.load_data()
ds_train = tf.data.Dataset.from_tensor_slices(x_train, y_train)

# define distributed strategy
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    # create and compile model
    model = get_model(arguments)
    model.compile(optimizer='adam‘,
        loss='sparse_categorical_crossentropy‘,
        metrics=['accuracy'])

# train model
model.fit(ds_train, epochs=5)
```

# TensorFlow – Distributed Examples (2)

- `MultiWorkerMirroredStrategy`

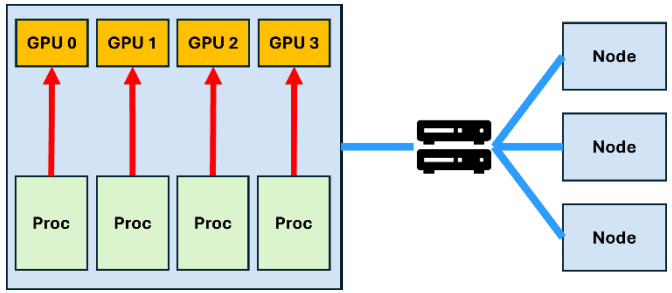



  - Further steps required
    - Define distributed strategy
    - Create and compile model under strategy scope
    - Limit device visibility (only see single GPU)
    - Set environment variable `TF_CONFIG`
  - TensorFlow will automatically take care of
    - Data sharding
    - Weight exchanges

```python
import tensorflow as tf

# load and preprocess dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), _ = mnist.load_data()
ds_train = tf.data.Dataset.from_tensor_slices(x_train, y_train)

# limit to local rank device
tf.config.set_visible_devices(devices[local_rank], 'GPU‘)

# Example: TF_CONFIG = '{"cluster": {"worker":
#                              ["host:12345", "host:23456"]},
#                       "task": {"type": "worker", "index": 0}}'

# define distributed strategy
strategy = tf.distribute.MultiWorkerMirroredStrategy()

with strategy.scope():
    # create and compile model
    model = get_model(arguments)
    model.compile(optimizer='adam‘,
        loss='sparse_categorical_crossentropy‘,
        metrics=['accuracy'])

# train model
model.fit(ds_train, epochs=5)
```

# TensorFlow – Distributed Examples (3)

- **Horovod**




  - Further steps required
    - Import Horovod
    - Init Horovod
    - Manual data sharding
    - Wrap optimizer
    - Unify model start weights

```python
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
import horovod.keras as hvd

hvd.init() # Horovod: initialize Horovod.

# load and preprocess dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), _ = mnist.load_data()
ds_train = tf.data.Dataset.from_tensor_slices(x_train, y_train)

# Horovod: need to manually shard dataset
ds_train = ds_train.shard(num_shards=hvd.size(), index=hvd.rank())

# create and compile model
model = get_model(arguments)
opt = hvd.DistributedOptimizer(Adam())
model.compile(optimizer=opt,
    loss='sparse_categorical_crossentropy‘,
    metrics=['accuracy'])

# train the model
model.fit(ds_train, epochs=5,
    callbacks=[hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
)
```

# How to execute on CLAIX?

## Approach 1: Containers (recommended)

- Virtualized OS encapsulating software with all dependencies to make it available on other machines
- On CLAIX: Predefined containers available

- Can be used by all users on the cluster
  - Reproducibility (development + production)
  - Less pressure on file system (only 1 copy)

- Once build they can't be modified
  - How to handle missing packages?

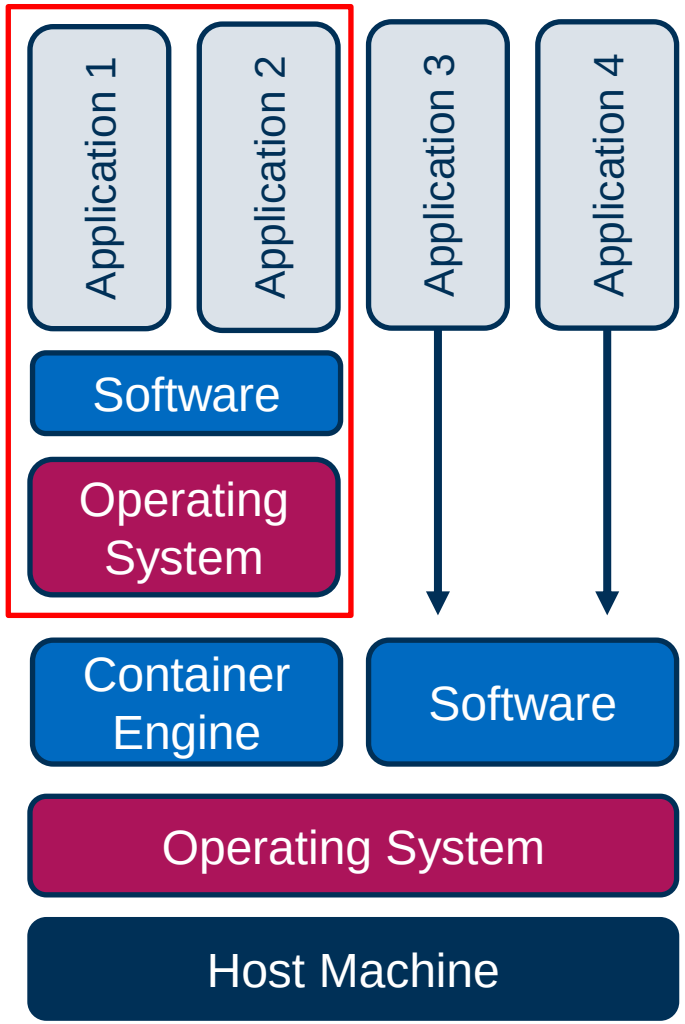
## Approach 2: Virtual Environments

- Different solutions available
  - Python virtualenv (pip)
  - conda-forge or ~~Anaconda~~
- Can also be used locally on your desktop

- Highly flexible
  - You can install and manage your package according to your needs

- Every user has its own installations
  - GBs and ~35k files per environment. High pressure on cluster file system / slow
  - Less reproducibility when working in teams
  - Hard to provide support

# More Information about Containers



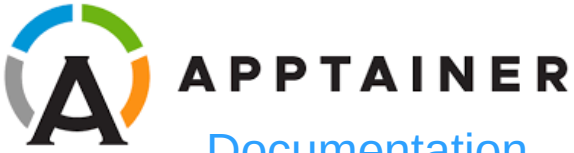

- The container engine (e.g., Docker or Apptainer), virtualizes parts of the operating system (process space, user namespace, network namespace)
- Containerized and native applications share the same host OS kernel. You can build it on one machine and use it somewhere else
- Negligible performance overhead when using containers

# Available Containers on CLAIX

- **Question:** What containers are available on CLAIX?

```
# show available software
module avail

------------------------------------------ Container Image Modules -------------------------------------------
datascience-notebook/6.4.8        PyTorch/nvcr-23.08-py3                    TensorFlow/nvcr-23.08-tf2-py3
datascience-notebook/7.0.3        PyTorch/nvcr-24.01-py3             (D)    TensorFlow/nvcr-24.01-tf2-py3 (D)
datascience-notebook/7.0.6 (D)    rapids/nvcr-22.02-cuda11.0-runtime
```

- Standard Apptainer containers for most common frameworks
  - Regular updates of container versions
  - Based on Docker images from DockerHub or NVIDIA




Documentation

- Support for user containers
  - Build and run your own containers
  - Possibility to convert Docker images to Apptainer images
  - Requests for standard containers → IT service desk (mailto:servicedesk@itc.rwth-aachen.de)

# Practical Examples / Scripts

- **Putting it all together**
  - SLURM and batch scripts
  - TensorFlow
  - Containers
  - …

**LIVE DEMO**

# Thank you!