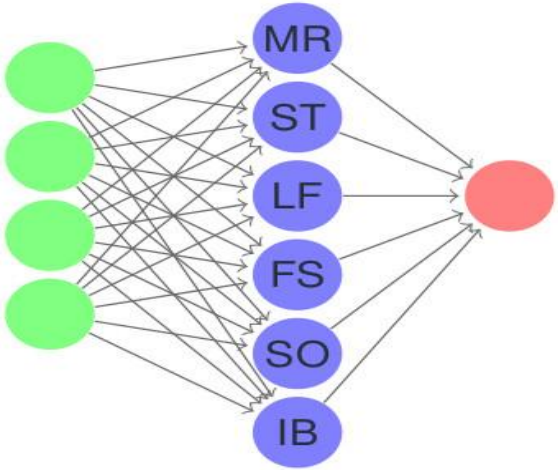
Introduction to Deep Learning and Tensorflow

Day 3 3/4

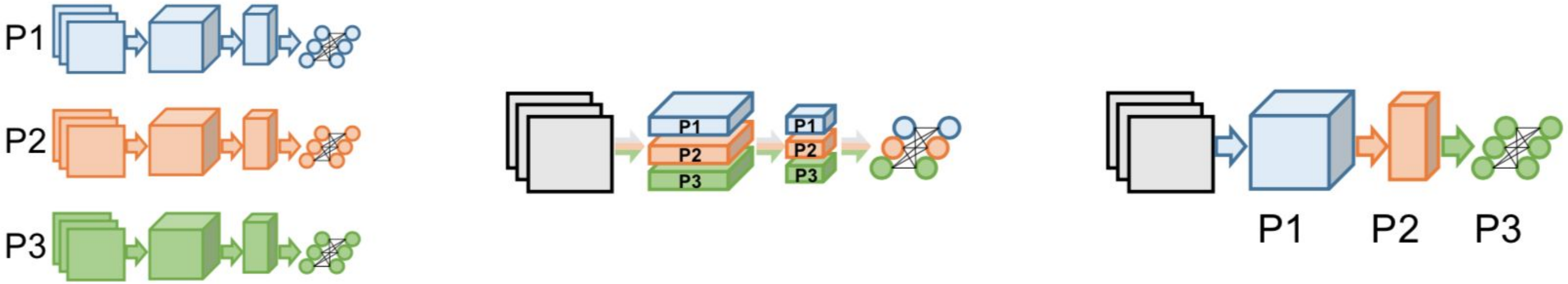


CINECA Roma - SCAI Department

Marco **R**orro
Luca **F**erraro
Sergio **O**rlandini
Stefano **T**agliaventi
Francesco **S**alvadore
Isabella **B**accarelli

Rome, 7th-9th Nov 2018

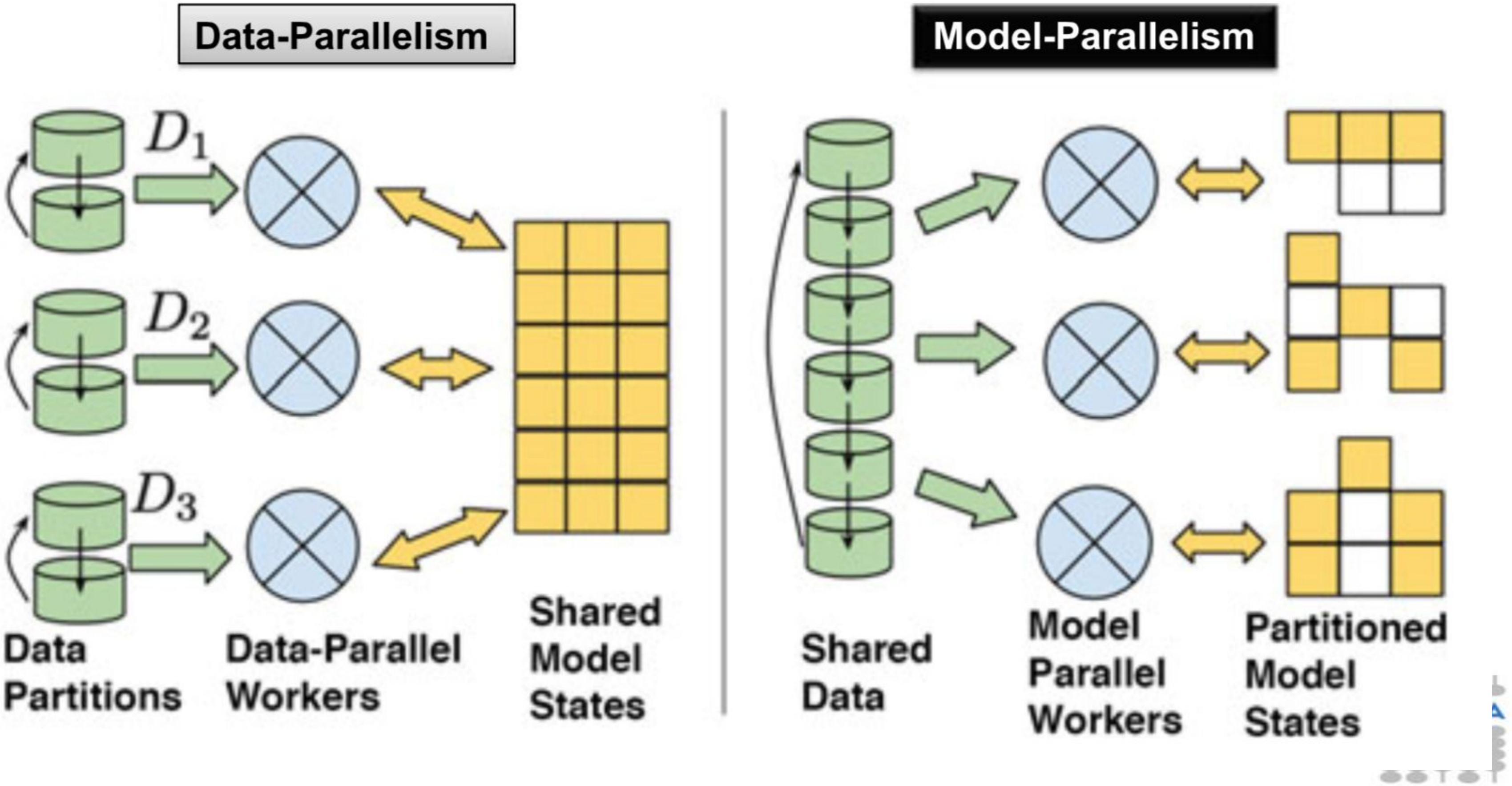
Neural Network concurrency



(a) Data Parallelism (b) Model Parallelism (c) Layer Pipelining

Tal Ben-Nun and Torsten Hoefler, Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, 2018,

Data Parallelism vs Model Parallelism



Hardware and Libraries

- It is not only a matter of computational power:
  - CPU (MKL-DNN)
  - GPU (cuDNN)
  - FGPA
  - TPU
- Input/Output matter
  - SSD
  - Parallel file system (if you run parallel algorithm)
- Communication and interconnection too, if you are running in distributed mode
  - MPI
  - gRPC +verbs (RDMA)
  - NCCL

Install TensorFlow from Source

```
[~]$ wget https://github.com/.../bazel-0.15.2-installer-linux-x86_64.sh
[~]$ ./bazel-0.15.2-installer-linux-x86_64.sh --prefix=...
[~]$ wget https://github.com/tensorflow/tensorflow/archive/v1.10.0.tar.gz
     ...
[~]$ python3 -m venv $TF_INSTALL_DIR
[~]$ source $TF_INSTALL_DIR/bin/activate
[~]$ pip3 install numpy wheel
[~]$ ./configure
     ...
[~]$ bazel build --config=mkl/cuda \
//tensorflow/tools/pip_package:build_pip_package
[~]$ bazel-bin/tensorflow/tools/pip_package/build_pip_package $WHEELREPO
[~]$ pip3 install $WHEELREPO/$WHL --ignore-installed
[~]$ pip3 install keras horovod ...
```

Input pipeline

If using accelerators like GPU, pipeline tha data load exploiting the CPU with the computation on GPU

The `tf.data` API helps to build flexible and efficient input pipelines

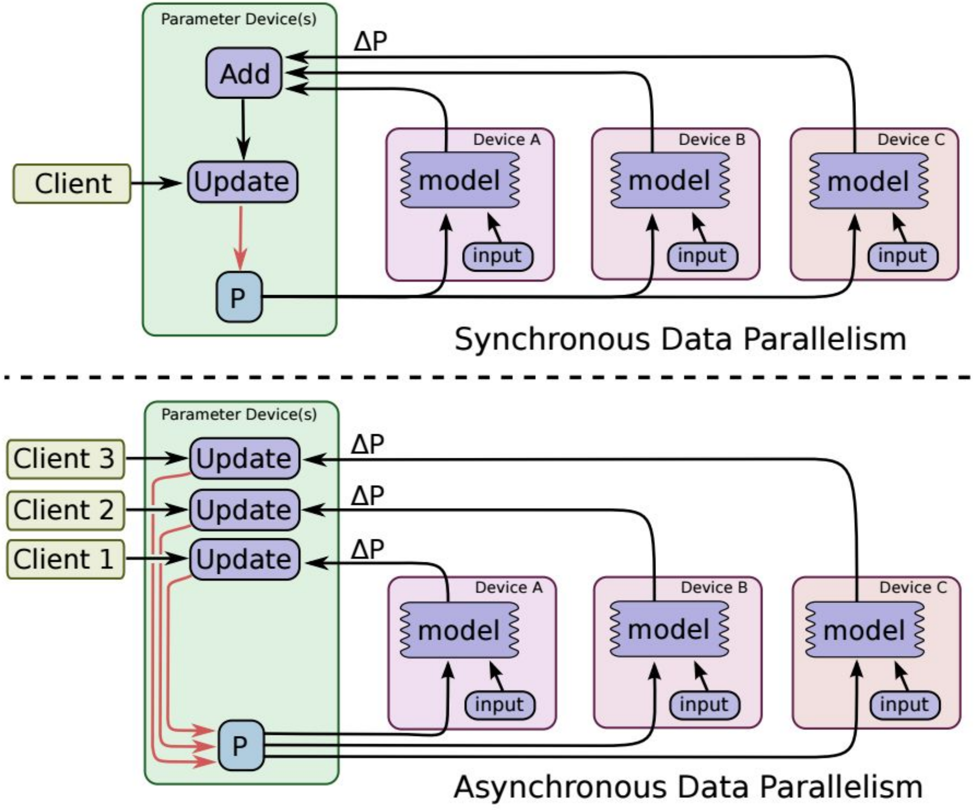
Optimizing for CPU

- Built from source with all of the instructions supported by the target CPU and the MKL-DNN option for Intel® CPU.
- Adjust thread pools
    - **`intra_op_parallelism_threads`:** Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool. (**`OMP_NUM_THREADS`**)
    - **`inter_op_parallelism_threads`**: All ready nodes are scheduled in this pool

```
config = tf.ConfigProto()
config.intra_op_parallelism_threads = 44
config.inter_op_parallelism_threads = 44
tf.session(config=config)
```

- The MKL is optimized for NCHW (default NHWC) data format and use the following variables to tune performance: **`KMP_BLOCKTIME`**, **`KMP_AFFINITY`**, **`OMP_NUM_THREADS`**

Synchronous and asynchronous data parallel training



[TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2016]

Distributed Tensorflow

```
#create a cluster from the parameter server and worker hosts.

cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})

#create a PS task

server = tf.train.Server(cluster, job_name="ps", task_index=0)

server.join()

#create a worker task

server = tf.train.Server(cluster, job_name="worker", task_index=0)

#build graph

with tf.device("/job:ps/task:0/cpu:0"):

    W = tf.Variable(...)
    opt = tf.train.GradientDescentOptimizer(.0001).minimize(loss)

    ...

With tf.device("/job:worker/task:0/gpu:0"):

    sess.run(opt)
```

Distributed Tensorflow with MPI + uber/horovod

```
import tensorflow as tf
import horovod.tensorflow as hvd

hvd.init() # Initialize Horovod

# Pin GPU to be used to process local rank (one GPU per process)
config = tf.ConfigProto()
config.gpu_options.visible_device_list =str( hvd.local_rank())

# Build model...
loss = ...
opt = tf.train.AdagradOptimizer(0.01 *  hvd.size())

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

# Add hook to broadcast variables from rank 0 to all other processes
during
# initialization.
hooks = [hvd.BroadcastGlobalVariablesHook(0)]
```

Distributed Tensorflow with MPI + uber/horovod

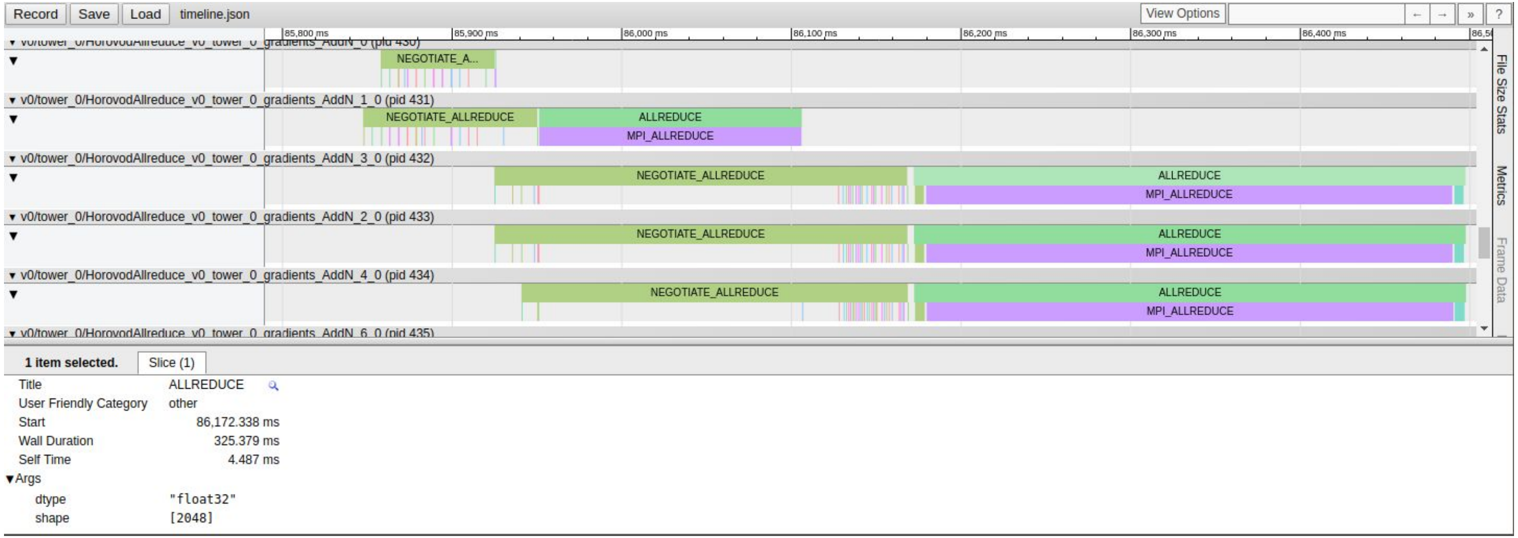
```
# Make training operation
train_op = opt.minimize(loss)

# Save checkpoints only on worker 0 to prevent other workers from
#corrupting them.
checkpoint_dir = '/tmp/train_logs' if  hvd.rank() == 0 else None

# The MonitoredTrainingSession takes care of session initialization,
# restoring from a checkpoint, saving to a checkpoint, and closing when
#done or an error occurs.
with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                       config=config,
                                       hooks=hooks) as mon_sess:
  while not mon_sess.should_stop():
    # Perform synchronous training.
    mon_sess.run(train_op)
```

Run and Analyze Horovod Performance

- To analyze Horovod performance:
  `[~]$ export HOROVOD_TIMELINE=/path/to/timeline.json`
- To tune the fusion buffer size:
  `[~]$ export HOROVOD_FUSION_THRESHOLD=33554432`
- To run: `[~]$ srun -n $NUM_GPUS python train.py`
- To visualize open the `timeline.json` in `chrome://tracing/`

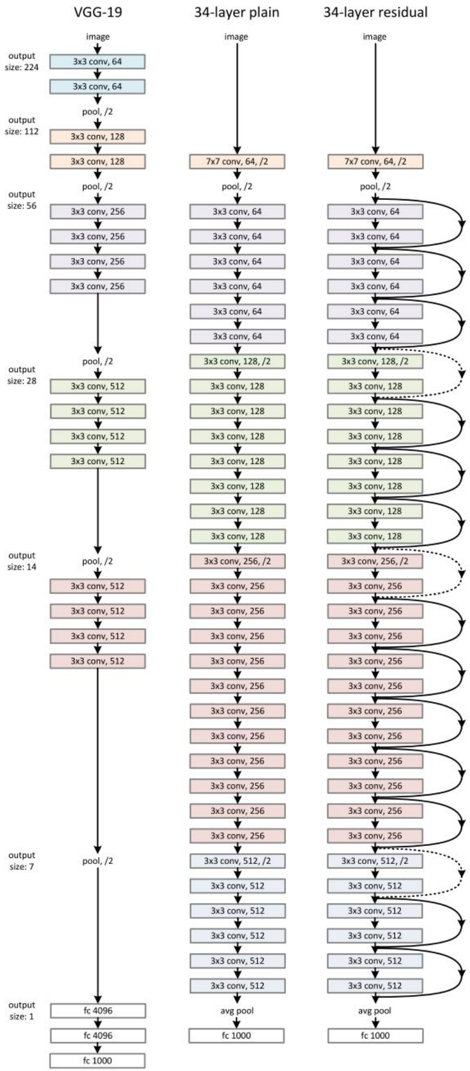


IMAGENET

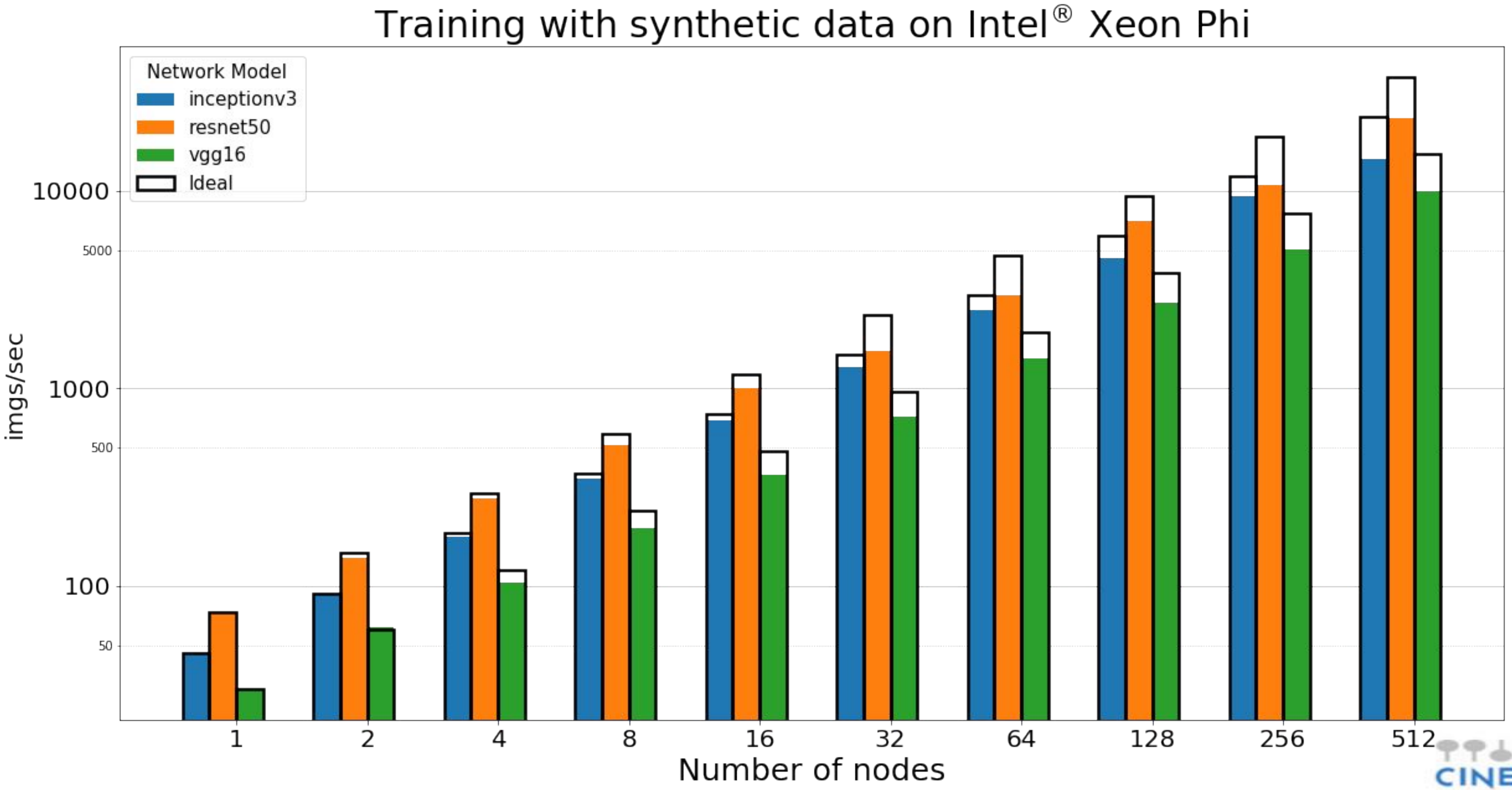
- 22000 classes 11M labeled image examples
- Reduced to 1000 classes and 1.4M images by taxonomy
- The smaller dataset has both fine and coarse-grained classes
- Synthetic version keeps size intact (224x224) but randomizes the content. Useful for benchmarking platforms and frameworks. Lifts I/O constraints.

Resnet50

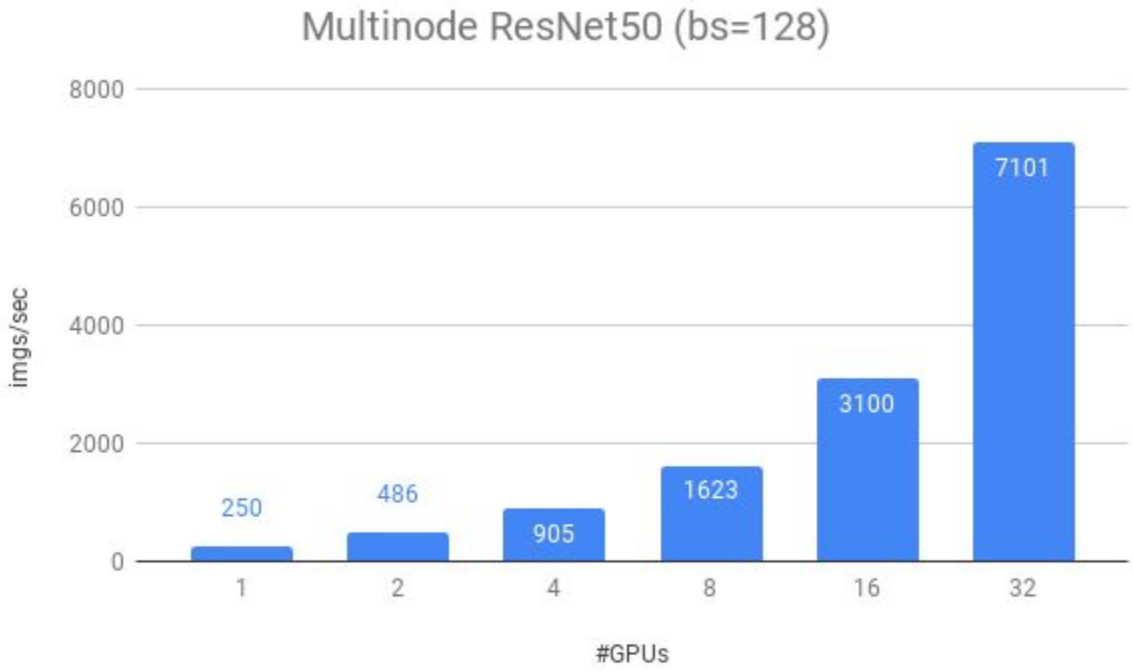
- Doesn't have "evil" pooling layers
- Uses batch normalisation
- Better pose information
- Higher accuracy models with less parameters than previously (let's say VGG)
- Good scaling behavior since it can be stochasticly trained
- State-of-the-art accuracy on ImageNet-1K: **75.3%**



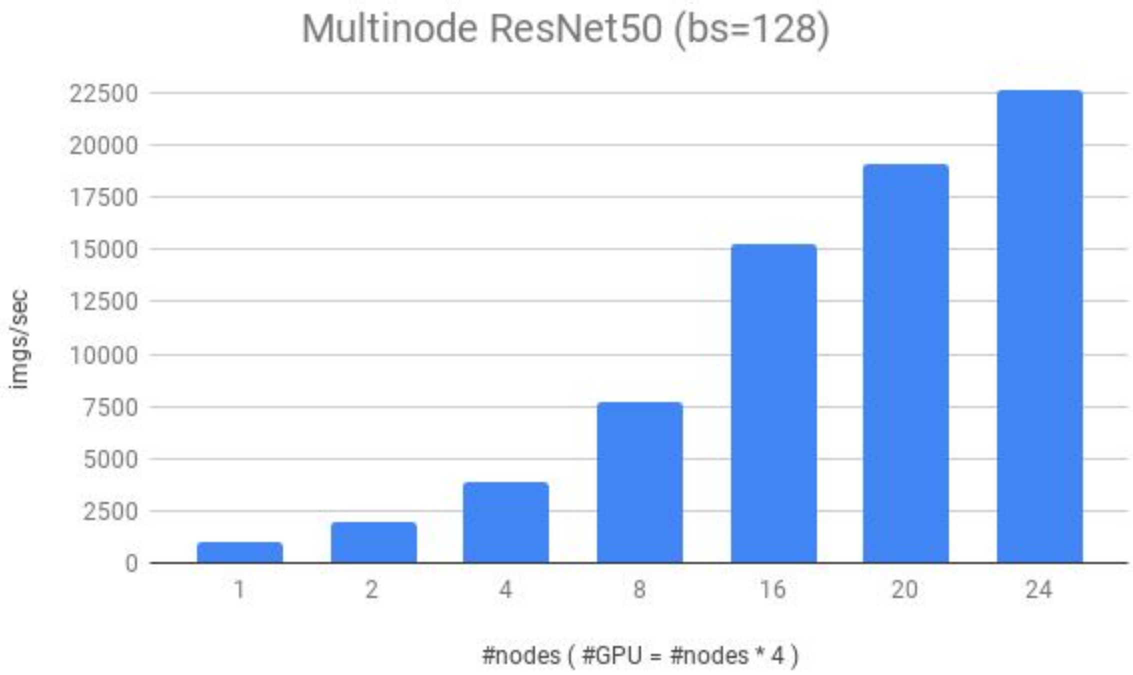
Horovod on Marconi



Horovod on DAVIDE



gRPC on DAVIDE



Q & A