Scaling Deep Learning Applications on Summit

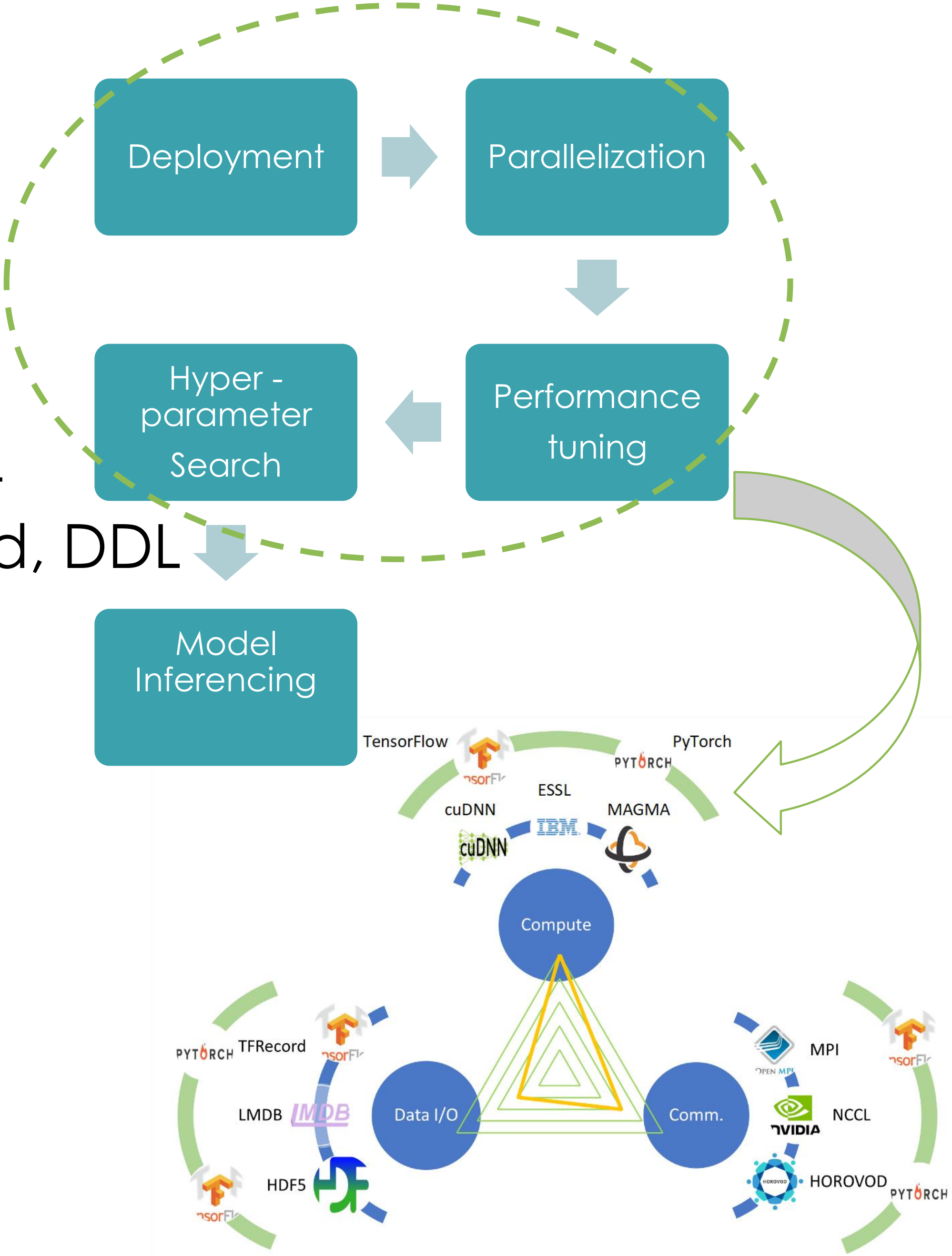
Junqi Yin

Advanced Data and Workflow

Oak Ridge Leadership Computing Facility

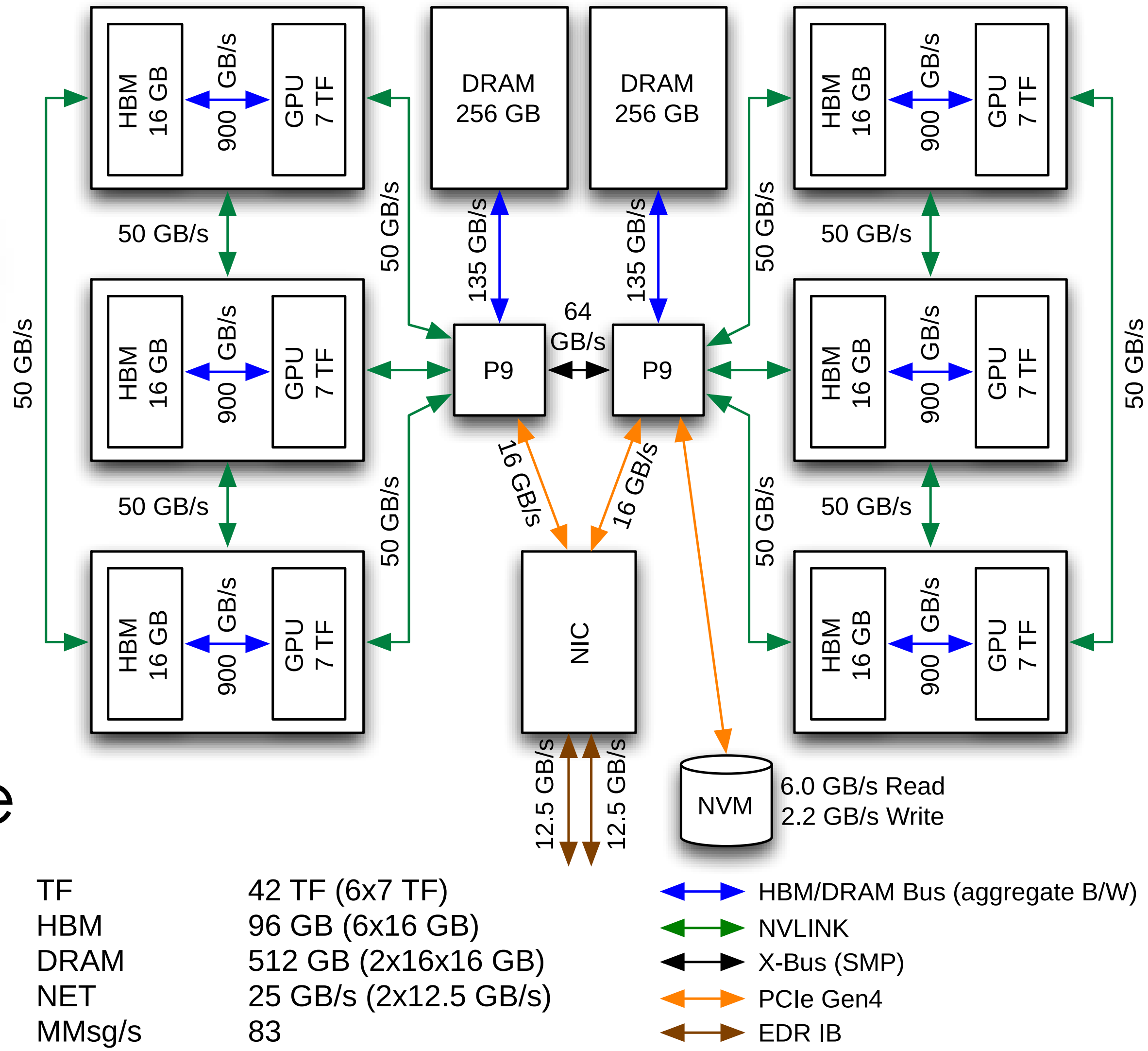
Outline

- DL on Summit overview
- Deployment and distributed DL
  - PyTorch: torch.distributed, Horovod, DDL
  - TensorFlow: distributed.Strategy, Horovod, DDL
- Performance tuning
  - Compute
  - I/O
  - Communication
- Hyperparameter search
- Model inferencing



Summit overview

- 27,648 V100 GPUs
- 3 ExaFlops in FP16 -> Compute
- 7 PB node local NVMe -> I/O
- NVLink2, EDR IB -> Comm.



TF	42 TF (6x7 TF)
HBM	96 GB (6x16 GB)
DRAM	512 GB (2x16x16 GB)
NET	25 GB/s (2x12.5 GB/s)
MMsg/s	83

HBM & DRAM speeds are aggregate (Read+Write).
All other speeds (X-Bus, NVLink, PCIe, IB) are bi-directional.

DL applications on Summit overview

- Full Summit DL
  - (1) "Exascale Deep Learning for Climate Analytics" (arXiv:1810.01993)
  - (2) "Exascale Deep Learning to Accelerate Cancer Research" (arXiv:1909.12291)
  - (3) "Exascale Deep Learning for Scientific Inverse Problems" (arXiv:1909.11150)

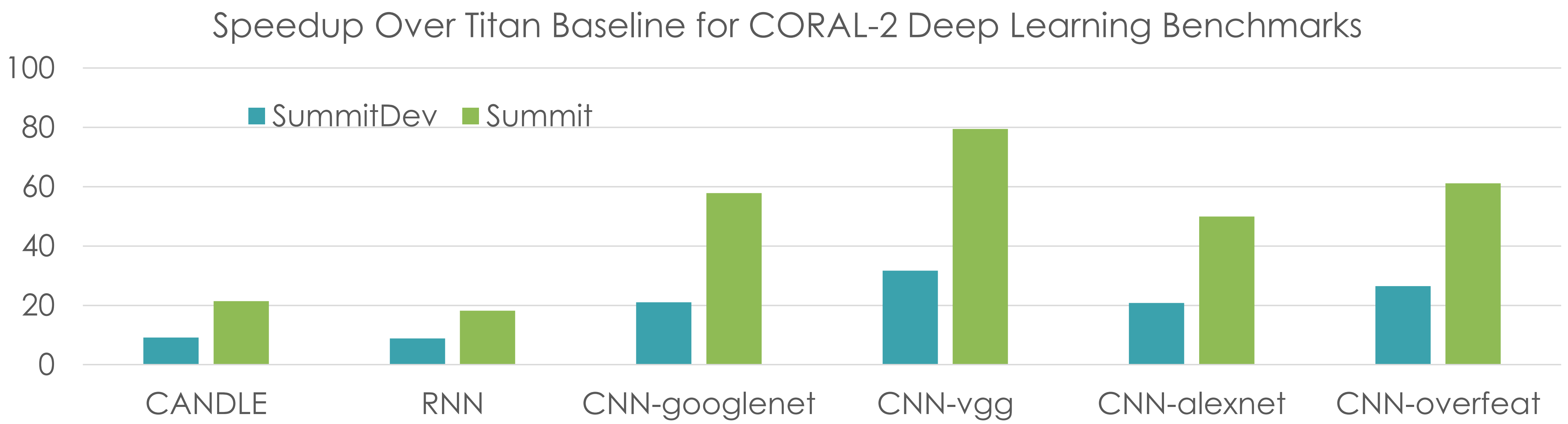
Application	Network	Sustained Performance (ExaFlops)	Peak Performance (ExaFlops)
(1) Climate	DeepLabV3+	0.999	1.13
(2) Medical	MENNDL	1.3	n/a
(3) Materials	Tiramisu variant	1.5	2.1

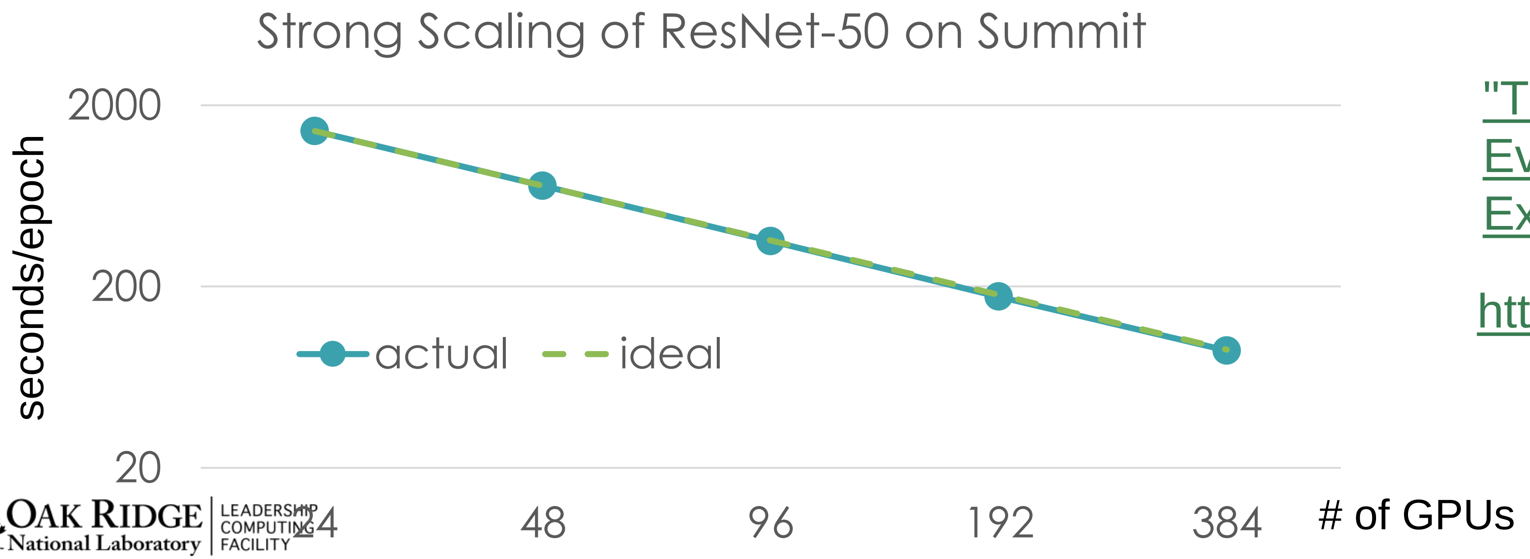
DL applications on Summit overview

- DL by domains and methods (growing list)

Domain\Method	Supervised Learning						Unsupervised Learning			Reinforcement Learning		Hyperparameter Search		
	Random Forest	MLP	CNN	RNN (LSTM)	Auto Encoder	GAN	Clustering	Evolution	Transformer	Q-Learning	Agent-based Learning	Evolution	RL-based	Transfer learning
Biophysics				✓	✓				✓	✓				
Chemistry			✓		✓									
Climate		✓	✓	✓		✓	✓							
Computer Science			✓		✓			✓			✓	✓	✓	✓
Fusion			✓											
Life Sciences	✓													
Material Sciences		✓	✓											
Medical Sciences			✓											
Nuclear Physics			✓	✓										
Turbulence			✓	✓										
Particle Physics			✓	✓		✓								

DL baselines on Summit: CORAL2 DL Benchmarks



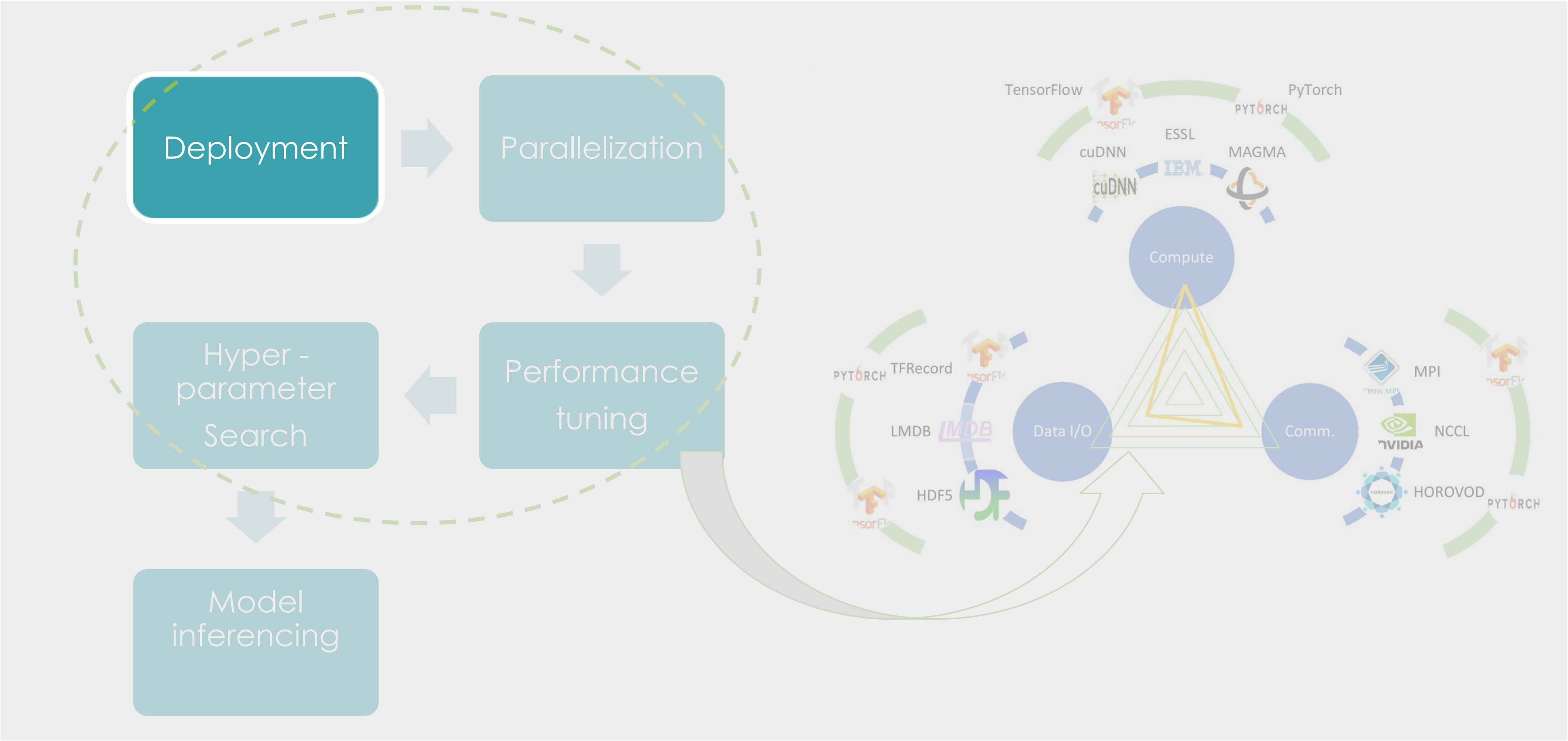


["The Design, Deployment, and Evaluation of the CORAL Pre-Exascale Systems", SC '18](https://asc.llnl.gov/coral-2-benchmarks/)

https://asc.llnl.gov/coral-2-benchmarks/

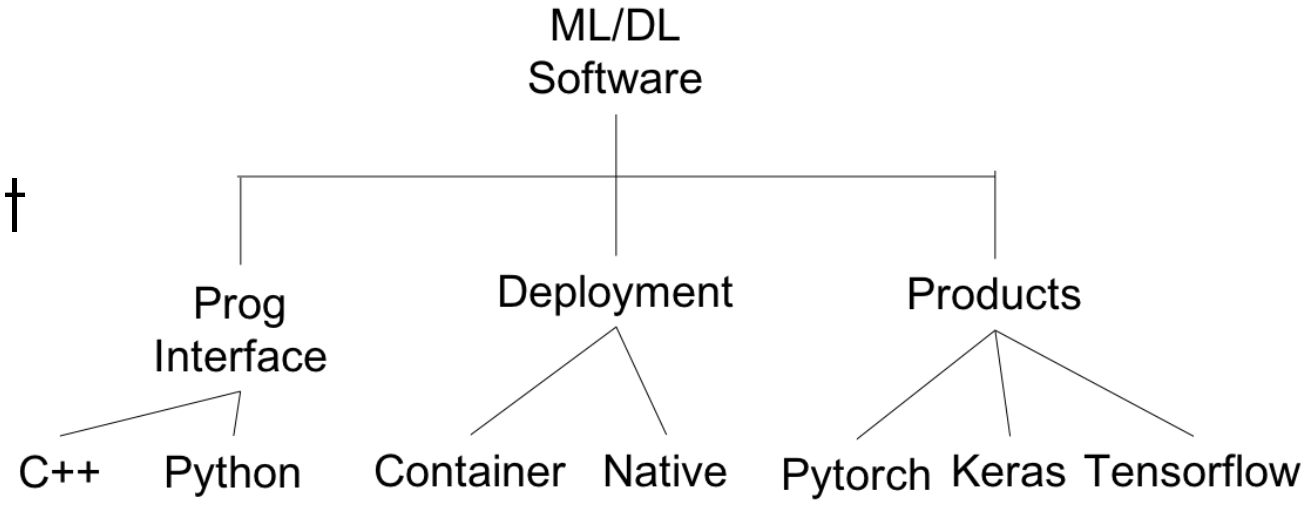
CORAL COLLABORATION
OAK RIDGE • ARGONNE • LIVERMORE

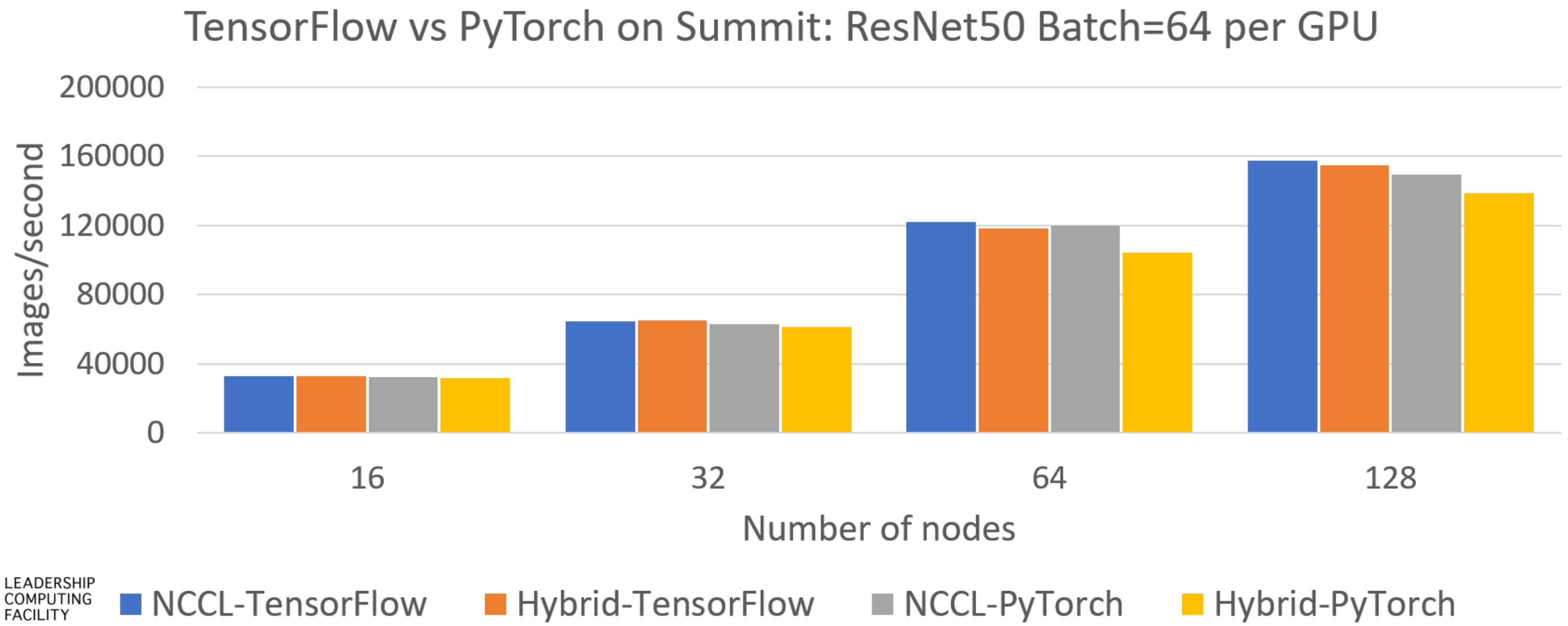
Outline


Deployment
Parallelization
Performance tuning
Hyper - parameter Search
Model inferencing
TensorFlow
PyTorch
ESSL
cuDNN
MAGMA
Compute
TFRecord
LMDB
HDF5
Data I/O
Comm.
MPI
NCCL
HOROVOD

DL Deployment considerations on Summit

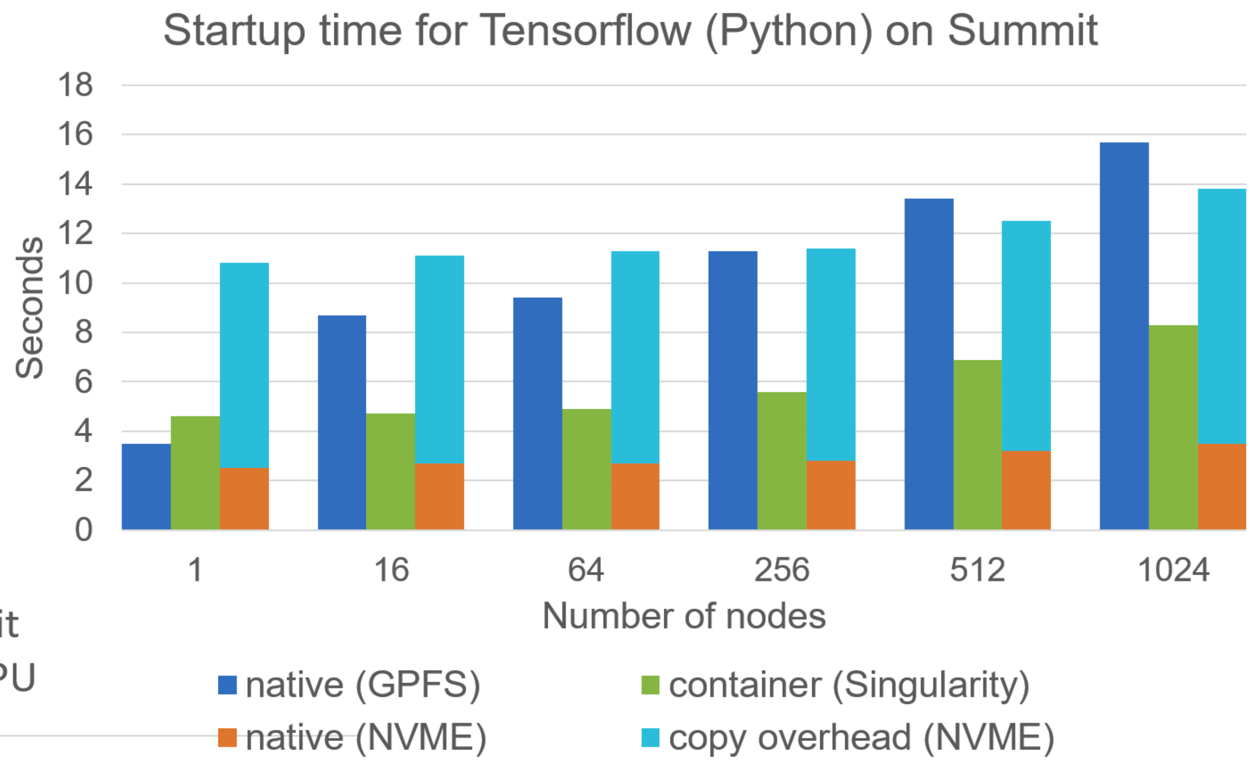
- ML/DL software deployment
- Framework comparisons
  - TensorFlow vs PyTorch

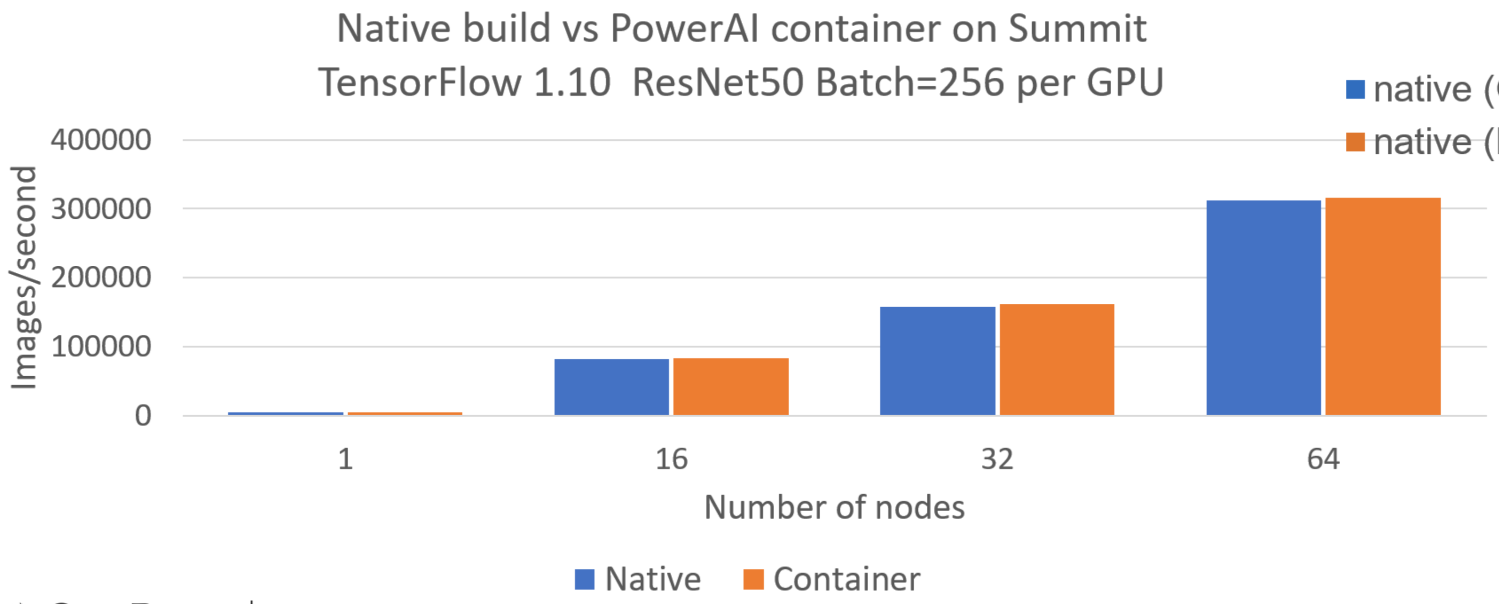




DL Deployment considerations on Summit

- Native vs Container
  - Impact of loading shared libs
  - Runtime performance

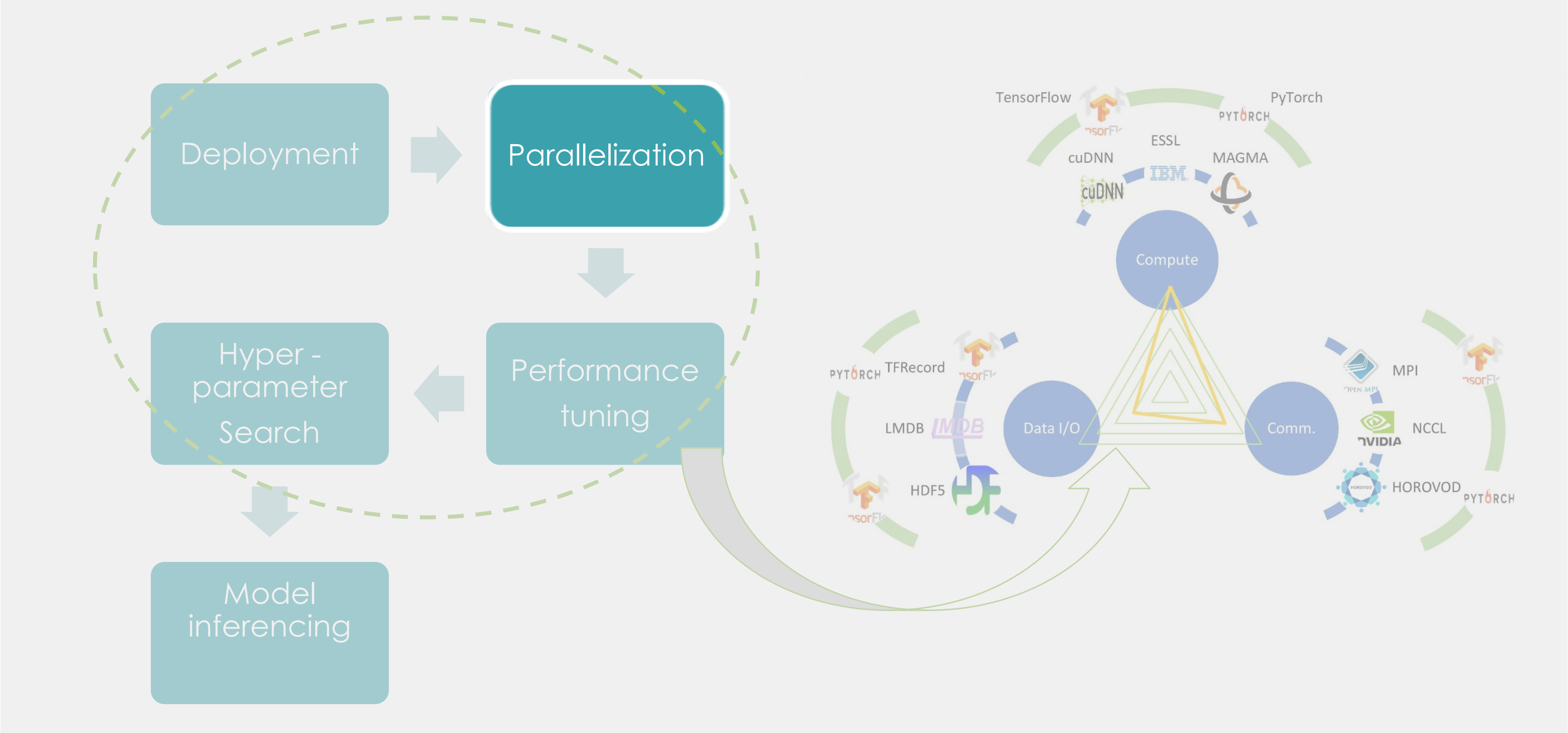




Container

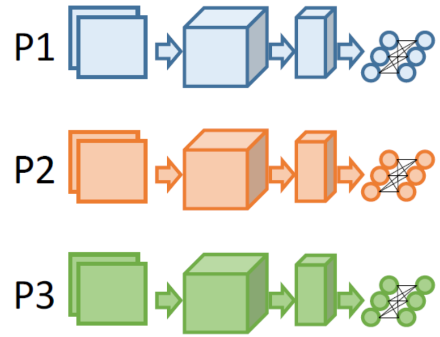
Pros	Cons
• Faster loading • Same runtime performance • Self-contain SW	• Build overhead • Lack of flexibility

Outline

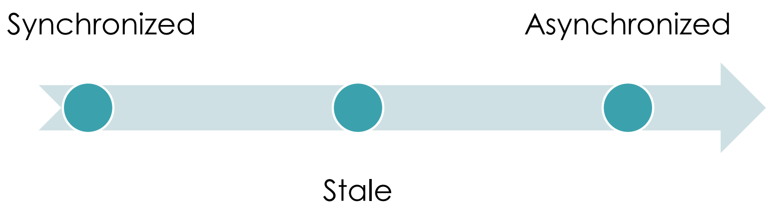

Deployment
Parallelization
Hyper - parameter Search
Performance tuning
Model inferencing
TensorFlow
PyTorch
ESSL
cuDNN
MAGMA
Compute
TFRecord
LMDB
HDF5
Data I/O
MPI
NCCL
HOROVOD
Comm.

Distributed deep learning: parallel scheme

- Data parallel



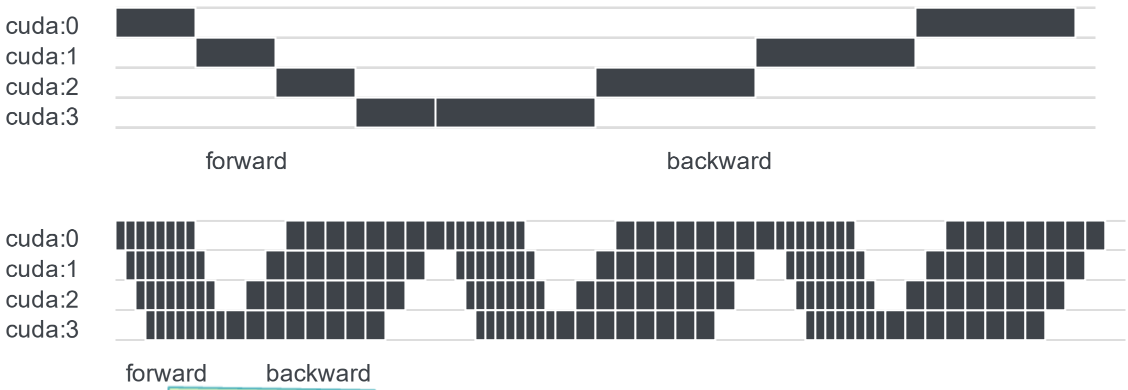
Model inconsistency:

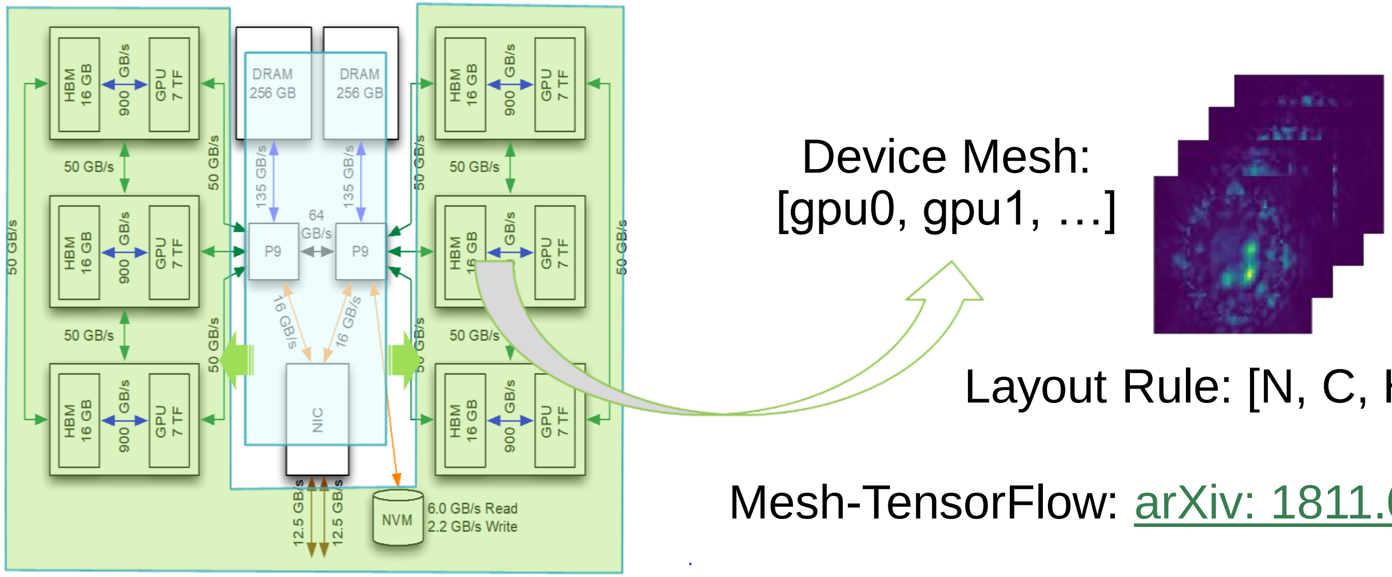


Review: [arXiv:1802.09941](https://arxiv.org/abs/1802.09941)

- Hybrid parallel

GPipe: [arXiv:1811.06965](https://arxiv.org/abs/1811.06965)





Device Mesh: [gpu0, gpu1, …]

Layout Rule: [N, C, H, W]

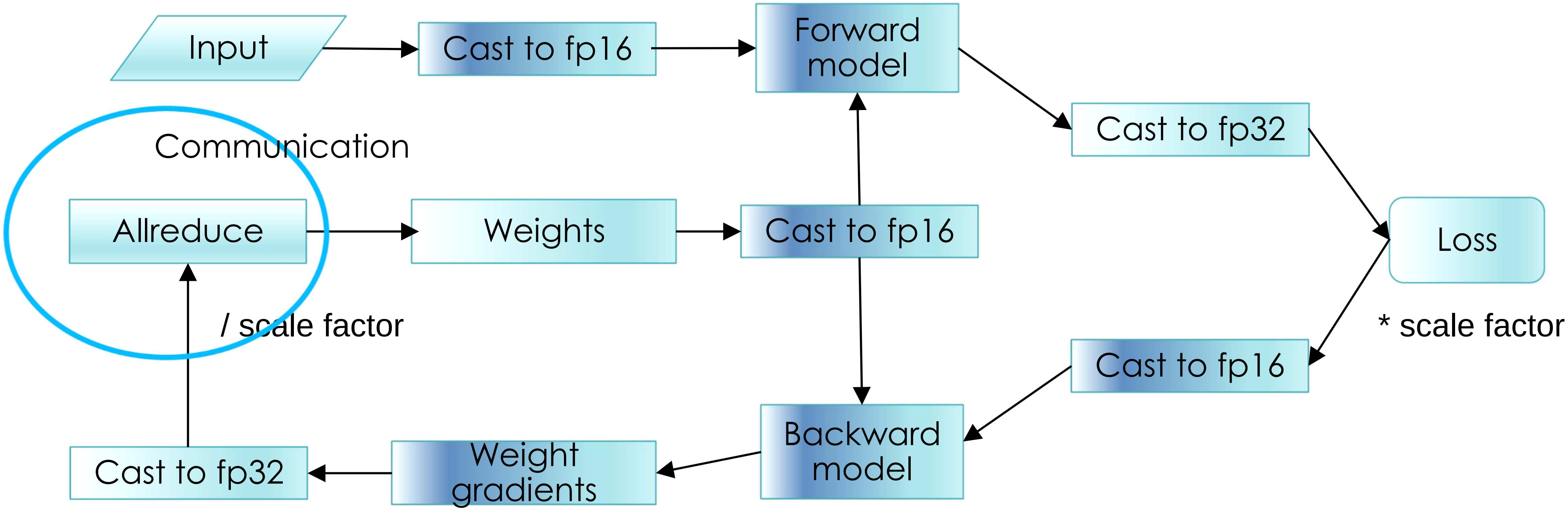
Mesh-TensorFlow: [arXiv: 1811.02084](https://arxiv.org/abs/1811.02084)

Difference in scaling up: DL VS simulation

- DL is a global optimization, changing scale (data parallel) -> changing solution space.
  - Hyperparameters often need to be re-tuned at different scale
- Scale in FLOPS ≠ Scale in time-to-solution (accuracy)
  - Tradeoff between samples/s and faster convergence
- High per-node FLOPS makes DL comm- and/or IO- bound at relatively small node count.
  - DL requires optimized comm (mainly all-reduce) and IO pipeline

Distribute your DL codes on Summit

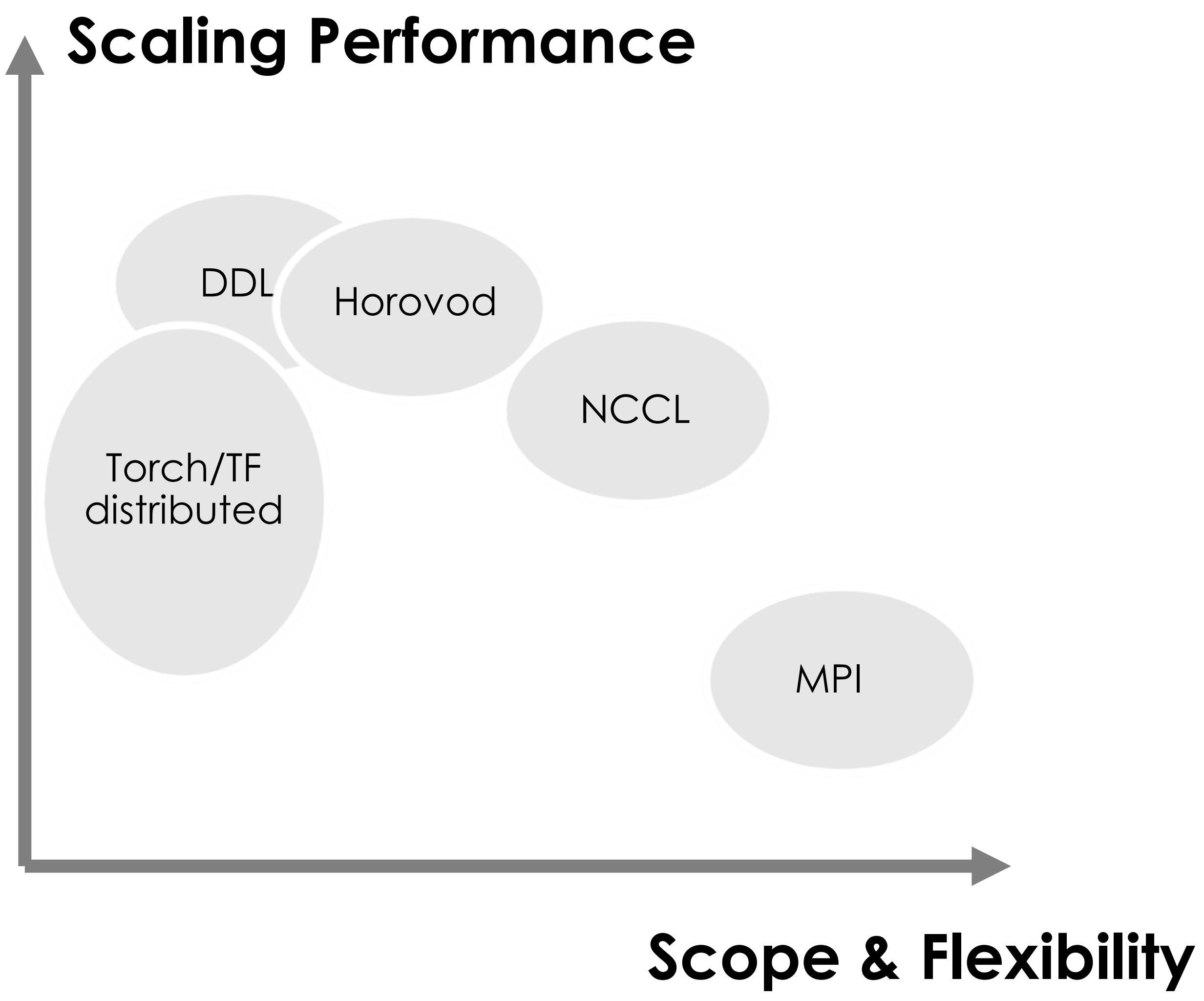
- Typical training flowchart
  - Allreduce of gradients in the backward propagation



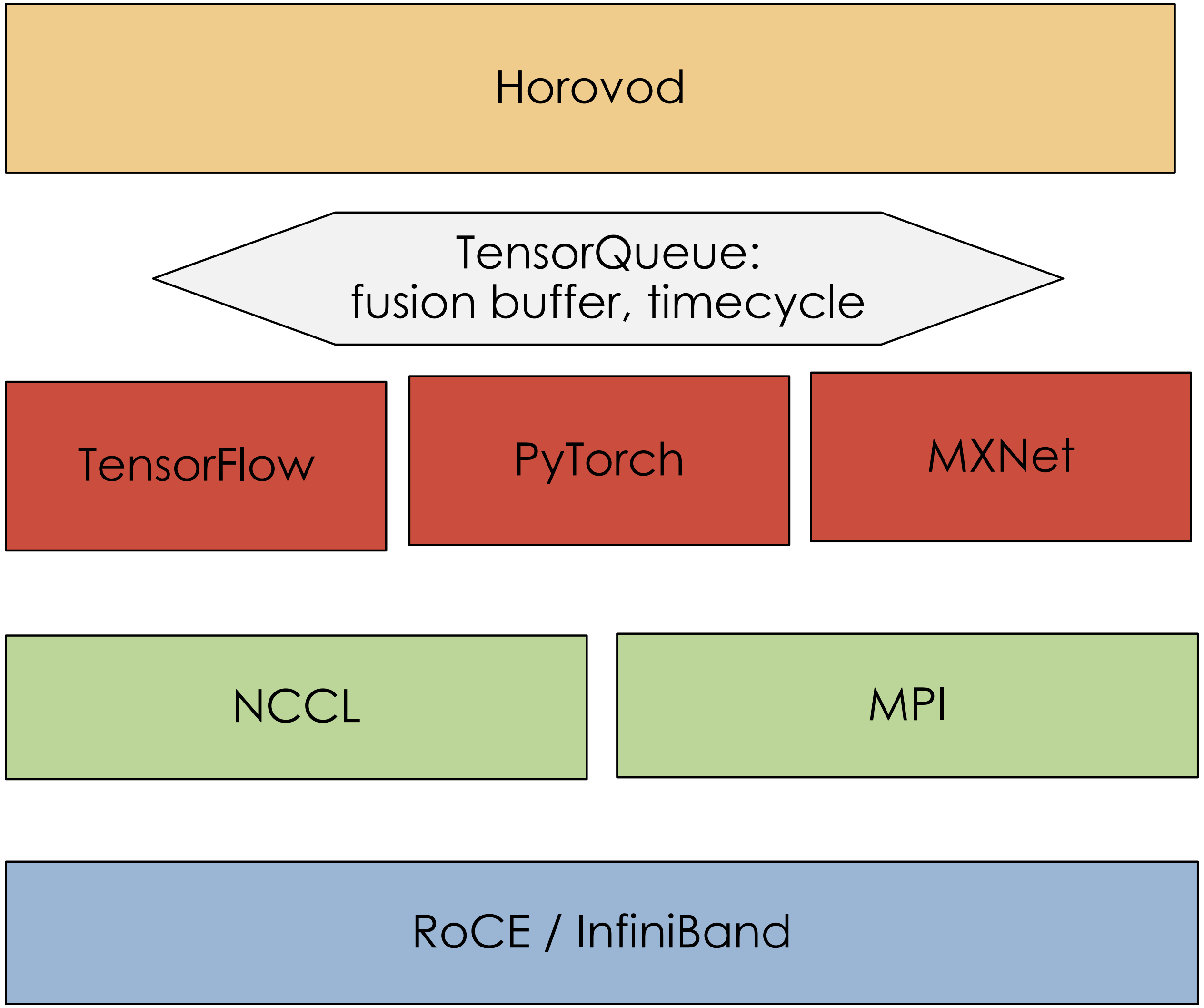
Distribute your DL codes on Summit

- Communication libraries
  - Low level: NCCL, MPI
  - High level: Hovorod, DDL

- Framework support
  - PyTorch:
    torch.distributed (NCCL or MPI)
  - TensorFlow:
    distributed.Strategy (NCCL)
  - 3rd Plugin: Horovod, DDL



Horovod: “mini-MPI” for distributed deep learning



- On top of MPI and NCCL
- Plugin support for TensorFlow, PyTorch, MXNet
- Tensor Fusion for better performances
- Key tuning parameters: fusion buffer and cycle time

Distribute your DL codes on Summit: PyTorch

- torch.distributed with NCCL and MPI backends: Initialization

```
world_size = int(os.environ['OMPI_COMM_WORLD_SIZE’])
world_rank = int(os.environ['OMPI_COMM_WORLD_RANK’])
local_rank = int(os.environ['OMPI_COMM_WORLD_LOCAL_RANK’])

if args.backend == 'nccl’:
    import subprocess
    get_master = "echo $(cat {} | sort | uniq | grep -v batch | grep -v login | head -1 )".format(os.environ['LSB_DJOB_HOSTFILE’])
    os.environ['MASTER_ADDR'] = str(subprocess.check_output(get_master, shell=True))[2:-3]
    os.environ['MASTER_PORT'] = "23456”
    os.environ['WORLD_SIZE'] = os.environ['OMPI_COMM_WORLD_SIZE’]
    os.environ['RANK'] = os.environ['OMPI_COMM_WORLD_RANK’]

dist.init_process_group(args.backend, rank=world_rank, world_size=world_size)
```

https://code.ornl.gov/olcf-analytics/summit/distributed-deep-learning-examples/tree/master/examples/pytorch

Distribute your DL codes on Summit: PyTorch

- torch.distributed with NCCL and MPI backends: Allreduce

```python
def average_gradients(model):
  """ Gradient averaging. """
  for param in model.parameters():
    dist.all_reduce(param.grad.data, op=dist.ReduceOp.SUM)
    param.grad.data /= world_size
```

```python
def benchmark_step():
  optimizer.zero_grad()
  output = model(data)
  loss = F.cross_entropy(output, target)
  loss.backward()
  if args.backend in ['nccl', 'mpi']:
    average_gradients(model)
  optimizer.step()
```

Allreduce

Distribute your DL codes on Summit: PyTorch

- Plugin distribution with Horovod and DDL:

```
if args.backend in ['horovod','ddl']:
    import horovod.torch as hvd
    hvd.init()

    # Horovod: pin GPU to local rank.
    torch.cuda.set_device(hvd.local_rank())

    # Horovod: wrap optimizer with DistributedOptimizer.
    optimizer = hvd.DistributedOptimizer(optimizer,
                                  named_parameters=model.named_parameters(),
                                  compression=compression)

    # Horovod: broadcast parameters & optimizer state.
    hvd.broadcast_parameters(model.state_dict(), root_rank=0)
    hvd.broadcast_optimizer_state(optimizer, root_rank=0)
```

Distribute your DL codes on Summit: TensorFlow

Build-in support for Data parallel

Framework \ Distribution	Single node	Multi node
TensorFlow	MirroredStrategy	MultiWorkerMirroredStrategy
PyTorch	DataParallel	DistributedDataParallel

- Easy of use and framework support
- High performance with NCCL backend
- Less flexible

Distribute your DL codes on Summit: TensorFlow

- Multi Worker Mirrored Strategy: TF_CONFIG setup

```
get_cnodes = "echo $(cat {} | sort | uniq | grep -v batch | grep -v
login)".format(os.environ['LSB_DJOB_HOSTFILE'])
cnodes = subprocess.check_output(get_cnodes, shell=True)
cnodes = str(cnodes)[2:-3].split(' ')
nodes_list = [c + ":2222" for c in cnodes]

# Add a port number # Get the rank of the compute node that is running on
index = int(os.environ['PMIX_RANK'])

# Set the TF_CONFIG environment variable to configure the cluster setting
os.environ['TF_CONFIG'] = json.dumps({
                         'cluster': {'worker': nodes_list },
                         'task'   : {'type': 'worker', 'index': index}
                         })
```

Distribute your DL codes on Summit: TensorFlow

- Multi Worker Mirrored Strategy: build-in support for tf.estimator

```
communication =
tf.distribute.experimental.CollectiveCommunication.NCCL

distribution_strategy =
tf.distribute.experimental.MultiWorkerMirroredStrategy(
communication=communication)


run_config = tf.estimator.RunConfig(
             train_distribute=distribution_strategy,
             session_config=session_config,
             save_checkpoints_secs=60*60*24,
             save_checkpoints_steps=None)
```

https://code.ornl.gov/olcf-analytics/summit/distributed-deep-learning-examples/tree/master/examples/tensorflow

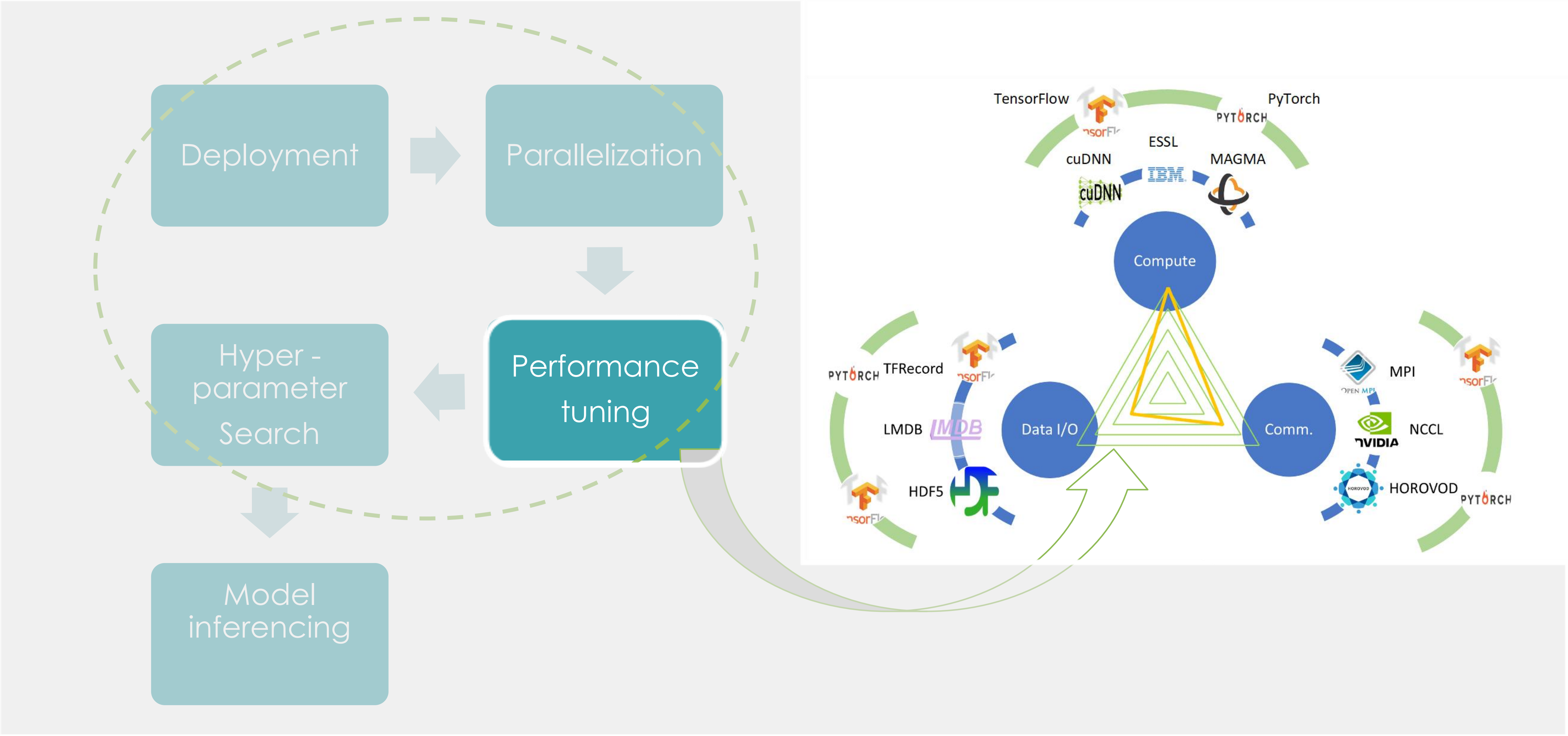
Always checkpointing

- It is relatively straightforward and cheap to checkpoint in DL.
- for data parallel, it is essentially the same as for single GPU.

```python
# Save checkpoint
if hvd.rank() == 0:
    state = {
            'model': model.state_dict(),
            'optimizer': optimizer.state_dict(),
    }
    torch.save(state, filepath)
```

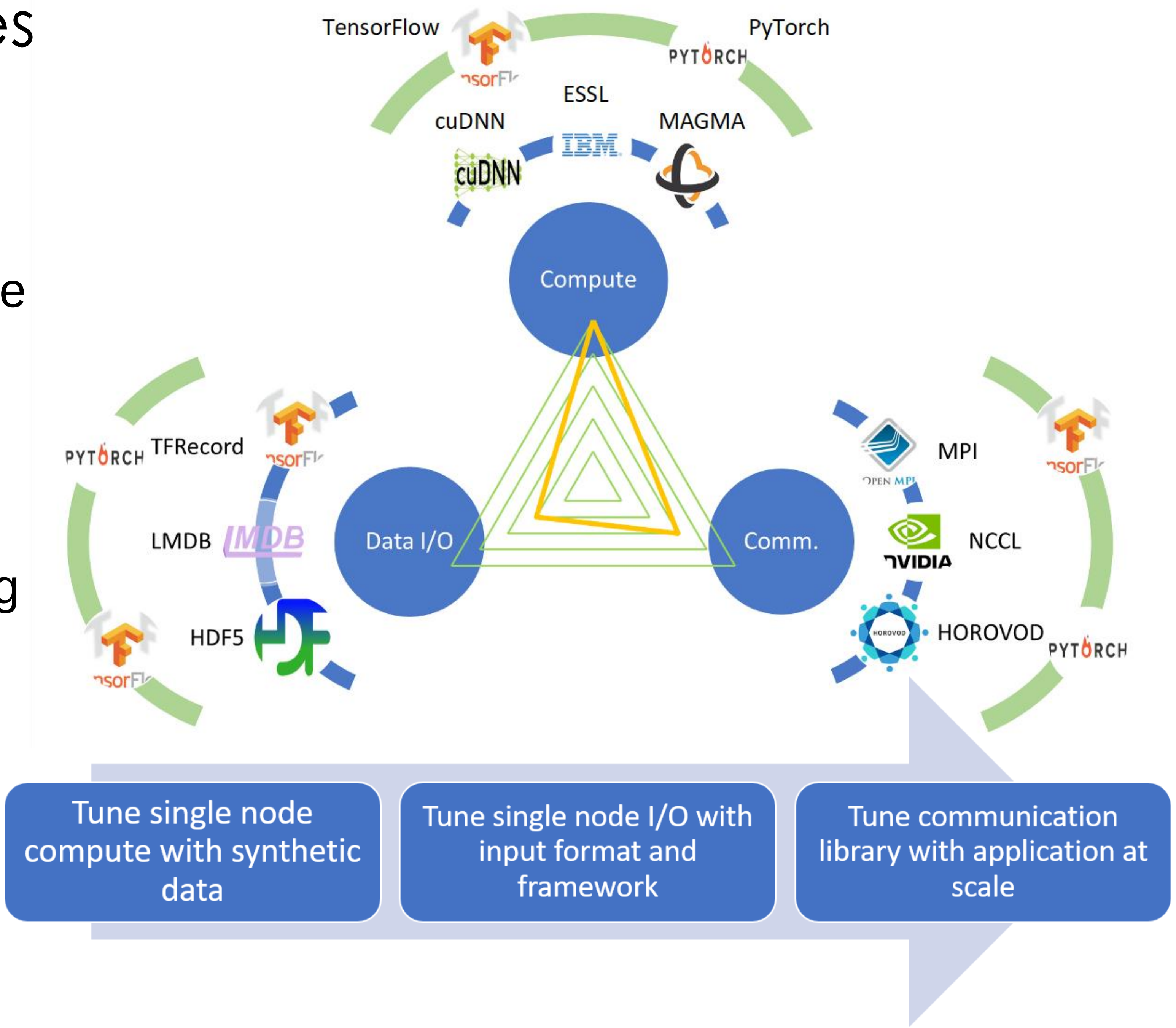
```python
# Load checkpoint
if hvd.rank() == 0:
    checkpoint = torch.load(filepath)
    model.load_state_dict(checkpoint['model'])
    optimizer.load_state_dict(checkpoint['optimizer'])

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(),
                         root_rank=0)
hvd.broadcast_optimizer_state(optimizer,
                              root_rank=0)
```

Outline


Deployment
Parallelization
Performance tuning
Hyper - parameter Search
Model inferencing
TensorFlow
PyTorch
ESSL
cuDNN
MAGMA
Compute
TFRecord
LMDB
HDF5
Data I/O
Comm.
MPI
NCCL
HOROVOD

Scaling strategies

1. Compute:
   – Tune on single node with synthetic data
2. I/O
   – Tune on NVMe with input pipelining
3. Communication
   – Tune at scale with comm. libs



Scaling considerations: Compute

- Use Tensor Cores (2 ~ 4x) 15 TFLOPS(FP32) vs 120 TFLOPS(FP16)
  - PyTorch: NVIDIA Apex plugin https://github.com/NVIDIA/apex

```
# Added after model and optimizer construction
model, optimizer = amp.initialize(model, optimizer, flags...)
# loss.backward() changed to:
with amp.scale_loss(loss, optimizer) as scaled_loss:
  scaled_loss.backward()
```

  - TensorFlow:

```
#Enable TF-AMP graph rewrite:
os.environ["TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE"] = "1"
#Enable Automated Mixed Precision:
os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
```

  - Verify on Tensorcore:

```
#Turn off Tensorcore:
os.environ["TF_DISABLE_CUDNN_TENSOR_OP_MATH"] = "0"
#nvprof: tensor_precision_fu_utilization to show TC utilization
```

Scaling considerations: Compute

- Use XLA (~ 1.5x for ResNet50)

```
#Enable XLA:
session_config.graph_options.optimizer_options.global_jit_level =
tf.OptimizerOptions.ON_1
```

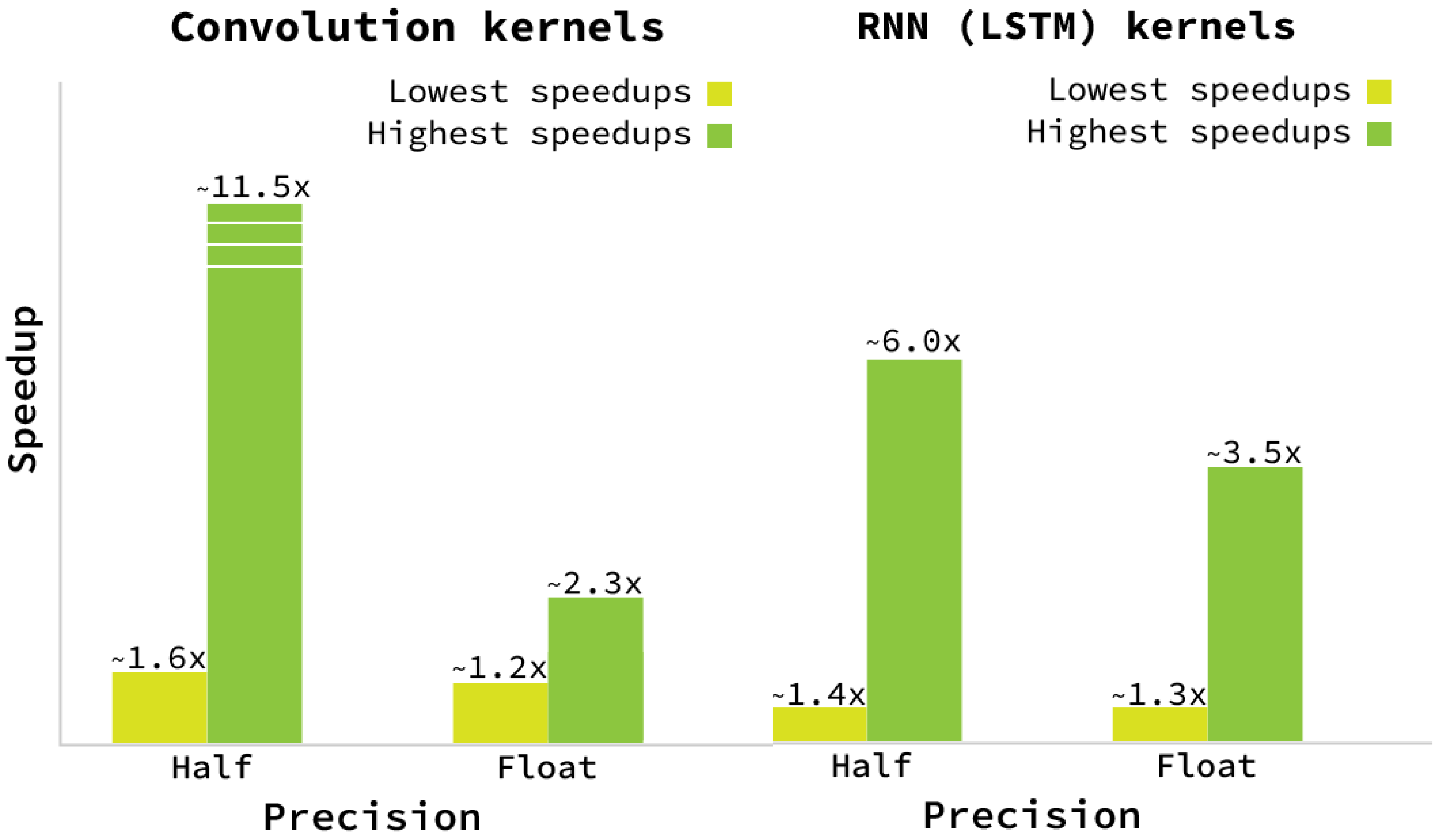
- Tune cuDNN algorithms (e.g. 7 implementations for conv)

```
#TensorFlow
os.environ['TF_CUDNN_USE_AUTOTUNE'] = '1'
#PyTorch
torch.backends.cudnn.benchmark = True
```

- Know your kernels (optimal scheduling policy)

Scaling considerations: Compute

- Benchmark kernels
  - CNN:
    kernel size, # of kernels, etc.
  - RNN:
    batch size, timesteps, etc.

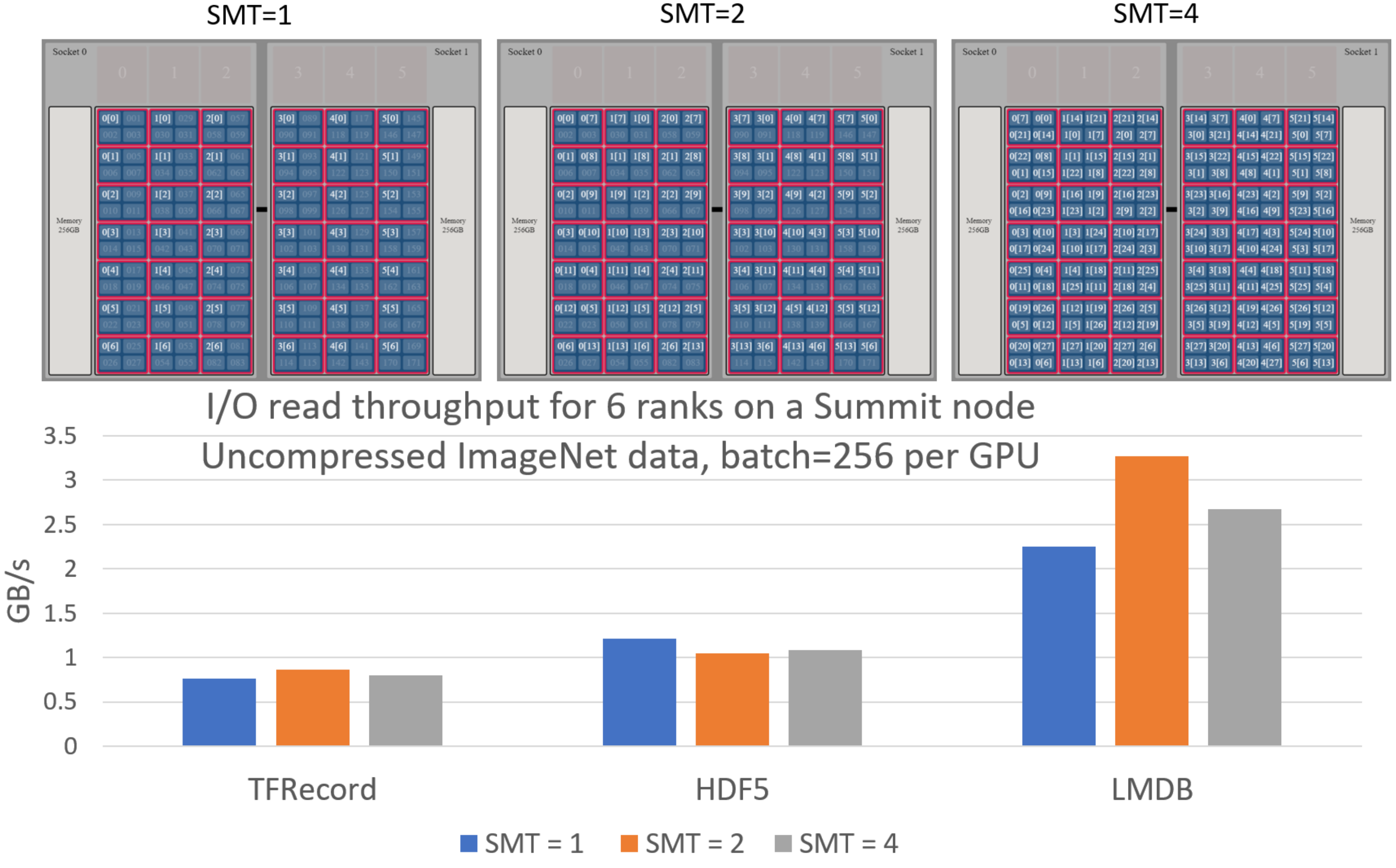


Scaling considerations: I/O

- Use node local NVMe

Device \Bandwidth	Full system run
GPUs processing	3*224*224*4* 1200*6*4608/10**12 ~ 20 TB/s
GPFS reading	2.5 TB/s
NVMe reading	6 GB/s * 4608 ~ 27 TB/s

- Use LMDB input format



Oak Ridge National Laboratory | Leadership Computing Facility

Scaling considerations: I/O

- CPU affinity settings (for pre-processing OPs)
  - Default binding:

```
-bash-4.2$ ps -eL -o pid,lwp,cpuid | grep 11707
 11707  11707    2
 11707  11746    1
 11707  11752    2
 11707  11757    3
 11707  11764    3
 11707  11769    0
 11707  11809    3
 11707  11810    1
 11707  11811    2
 11707  11812    3
```

  -- Correct binding (numactl + OMP_PLACE):

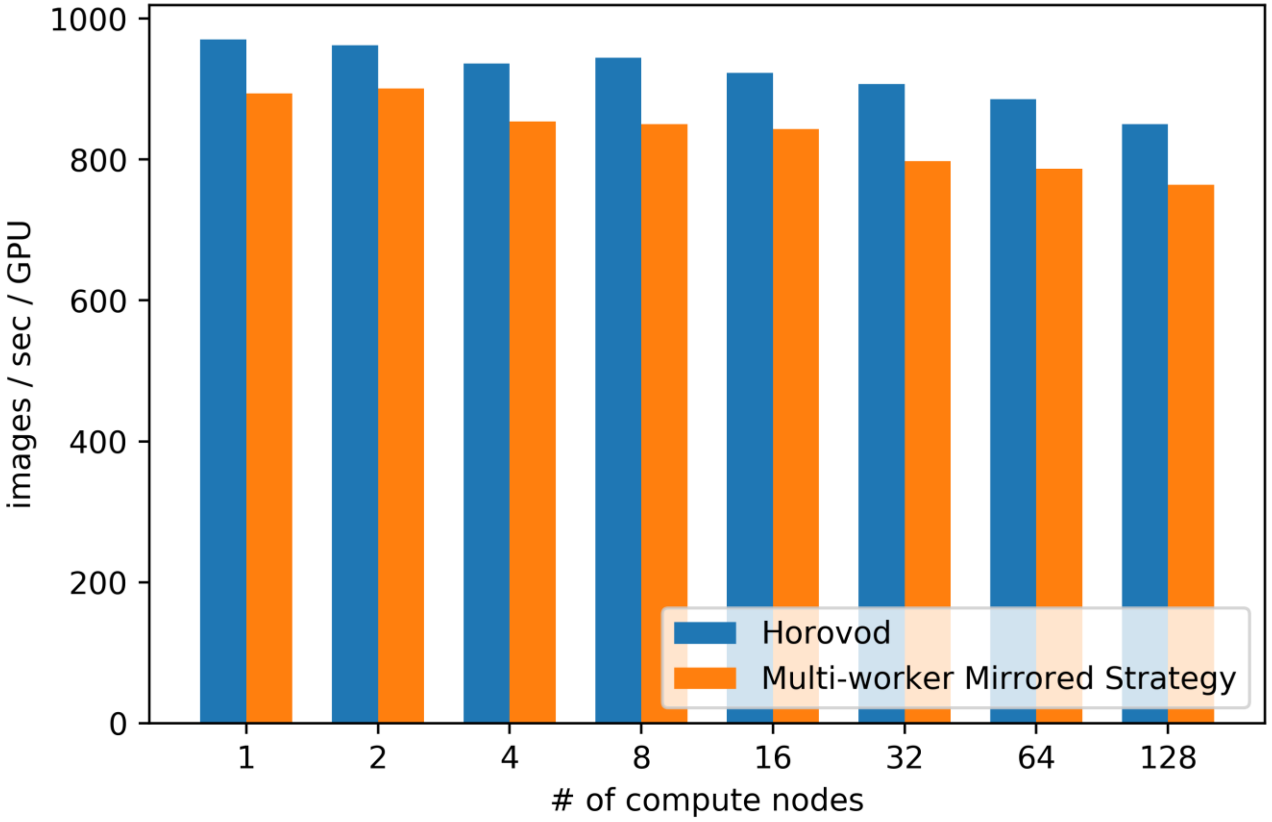
```
# for SMT4
case $(($PMIX_RANK%6)) in
[0])
export PAMI_IBV_DEVICE_NAME=mlx5_0:1
export OMP_PLACES={0:28}
numactl --physcpubind=0-27 --membind=0 $APP
;;
[1])
export PAMI_IBV_DEVICE_NAME=mlx5_1:1
export OMP_PLACES={28:28}
numactl --physcpubind=28-55 --membind=0 $APP
……
```

- Use pre-processing pipeline https://www.tensorflow.org/guide/data_performance : staging, prefetch, parallel_interleave, etc

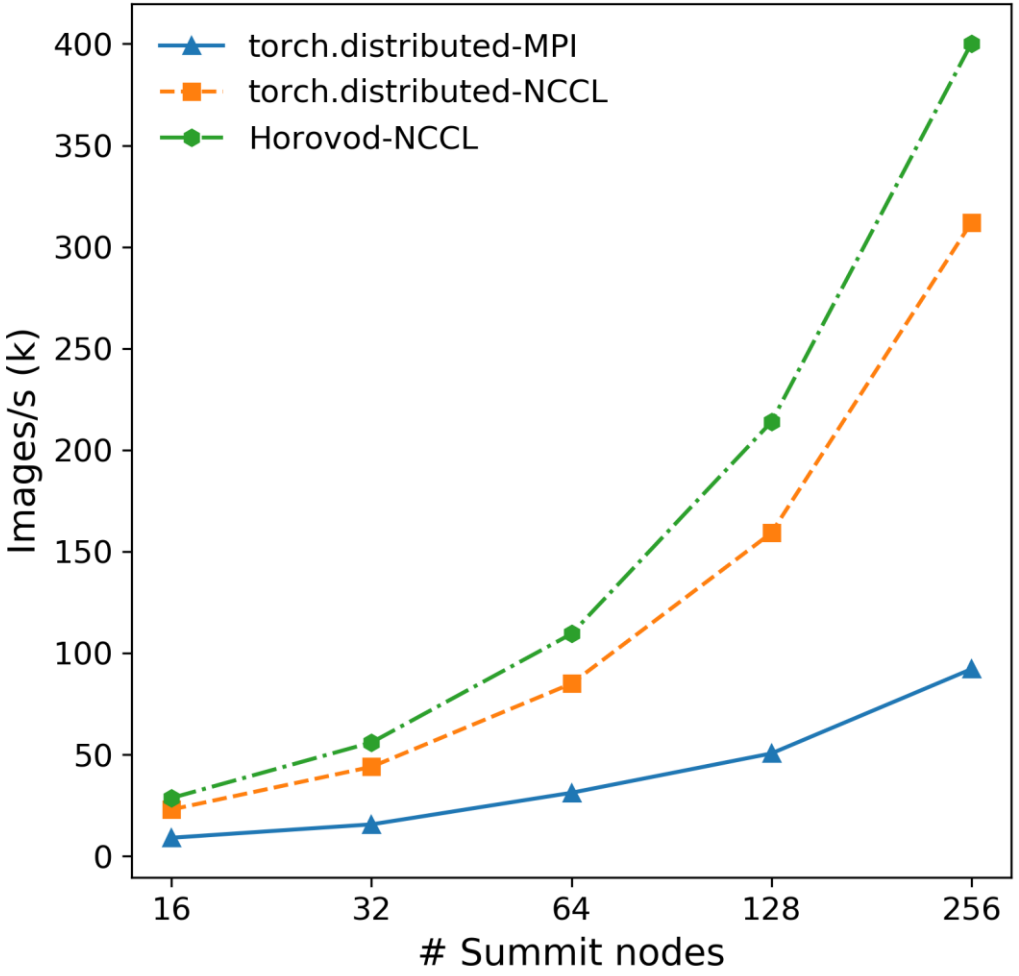
Scaling considerations: Communication

- Use Horovod with NCCL backend

TensorFlow



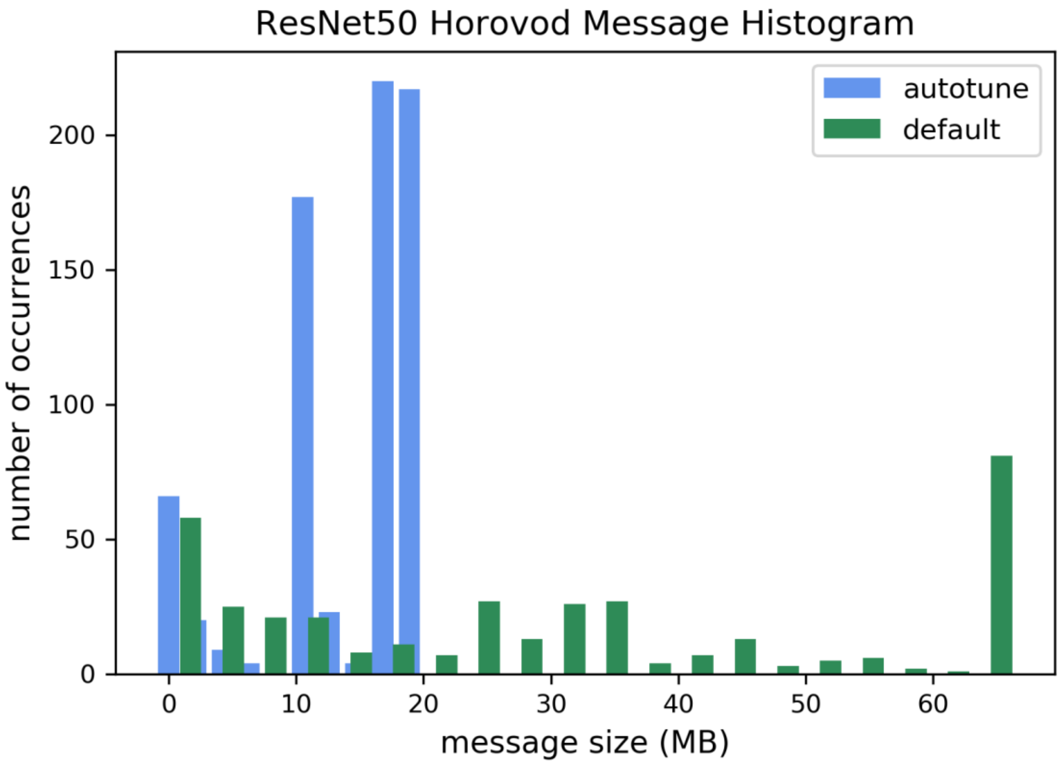
PyTorch



Scaling considerations: Communication

- Tune Horovod parameters
  - Key knobs: HOROVOD_CYCLE_TIME, HOROVOD_FUSION_THRESHOLD

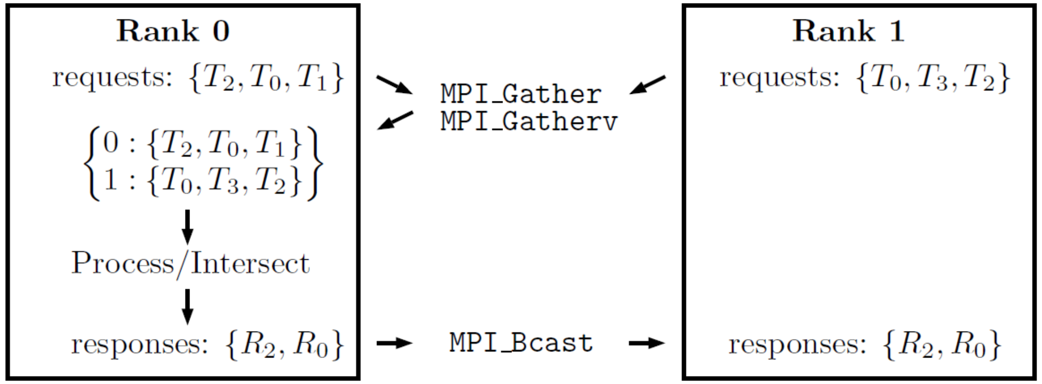
```
#Horovod autotuner
export HOROVOD_AUTOTUNE=1
export HOROVOD_HIERARCHICAL_ALLGATHER=0
export HOROVOD_HIERARCHICAL_ALLREDUCE=0
export NCCL_DEBUG_SUBSYS=COLL
```



Scaling considerations: Communication

- Add Bit-Allreduce to Horovod https://arxiv.org/abs/1909.11150

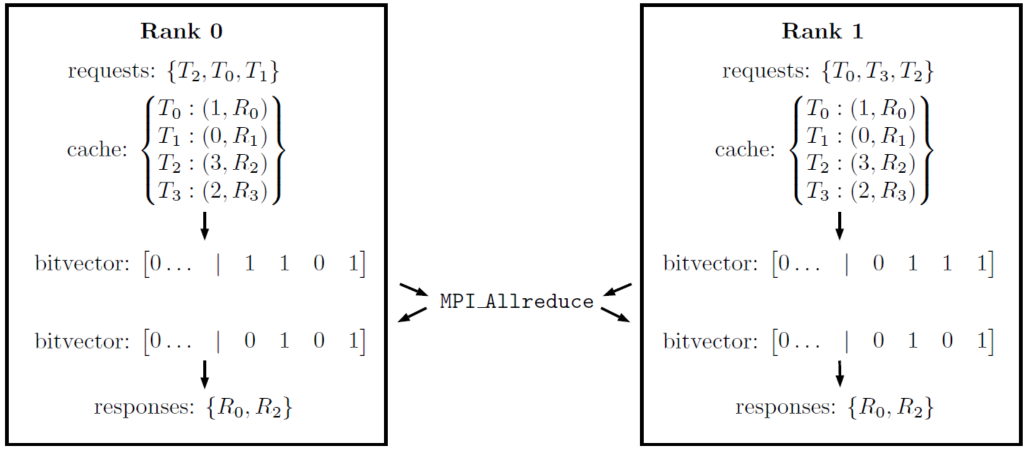
Original coordination strategy:

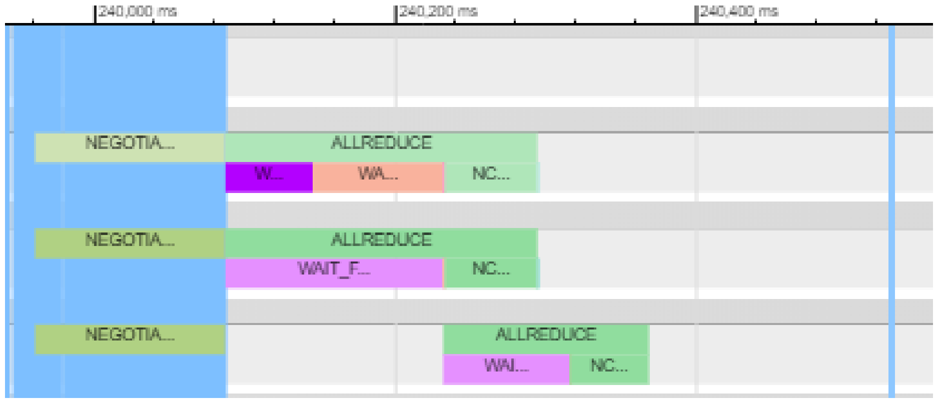


NEGOTIATE_ALLREDUCE > ALLRECUE

Name	Wall Duration	Self time
NEGOTIATE_ALLREDUCE	9,235.223 ms	9,235.223 ms
ALLREDUCE	839.756 ms	24.251 ms
INIT_NCCL	678.165 ms	678.165 ms
MPI_ALLREDUCE	82.068 ms	82.068 ms
NCCL_ALLGATHER	17.802 ms	17.802 ms
NCCL_REDUCESCATTER	11.042 ms	11.042 ms
MEMCPY_OUT_HOST_BUFFER	6.845 ms	6.845 ms
MEMCPY_IN_HOST_BUFFER	5.763 ms	5.763 ms
MEMCPY_OUT_FUSION_BUFFER	4.305 ms	4.305 ms

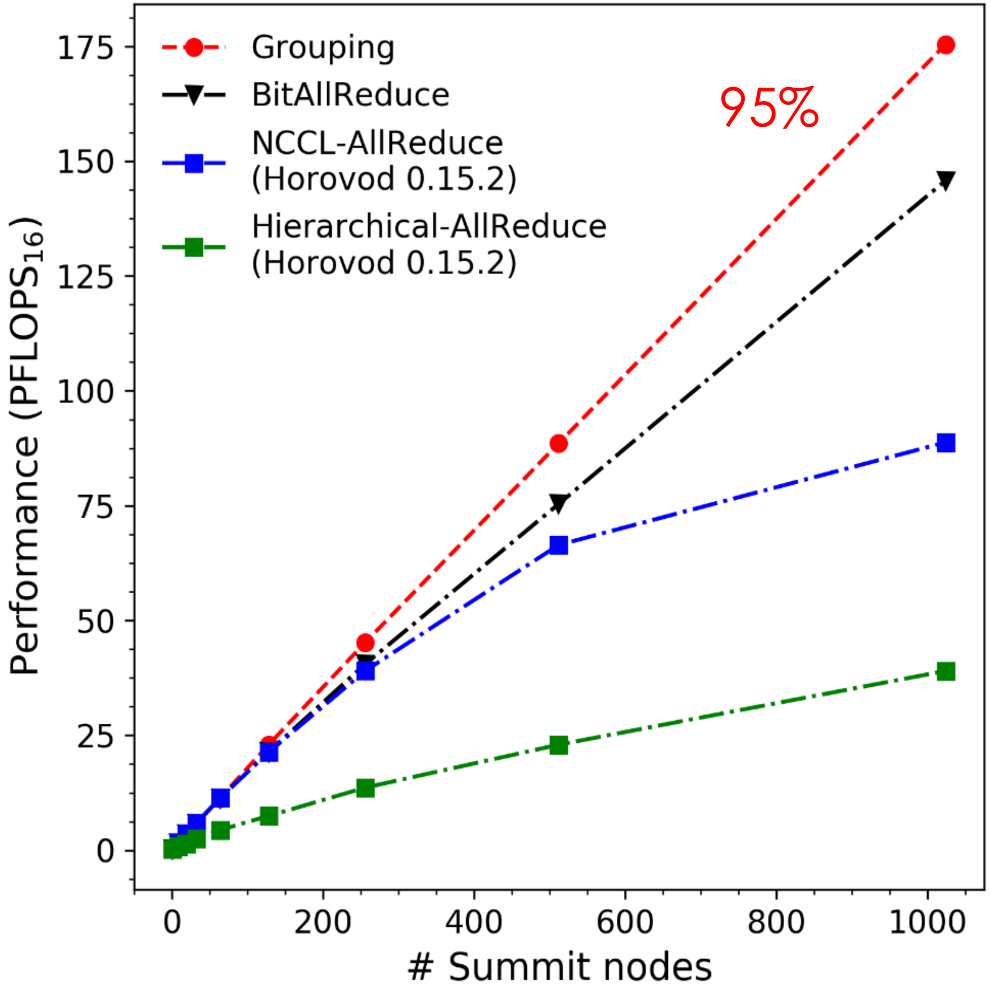
Improved coordination strategy:



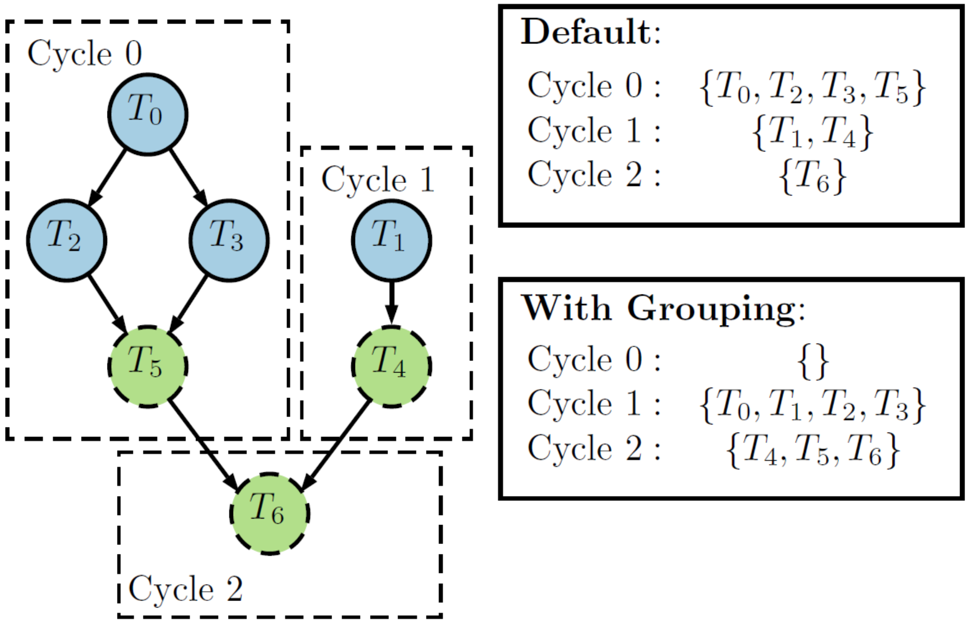


Scaling considerations: Communication

- Add Grouping to Horovod https://arxiv.org/abs/1909.11150



Map neural net topology to message group

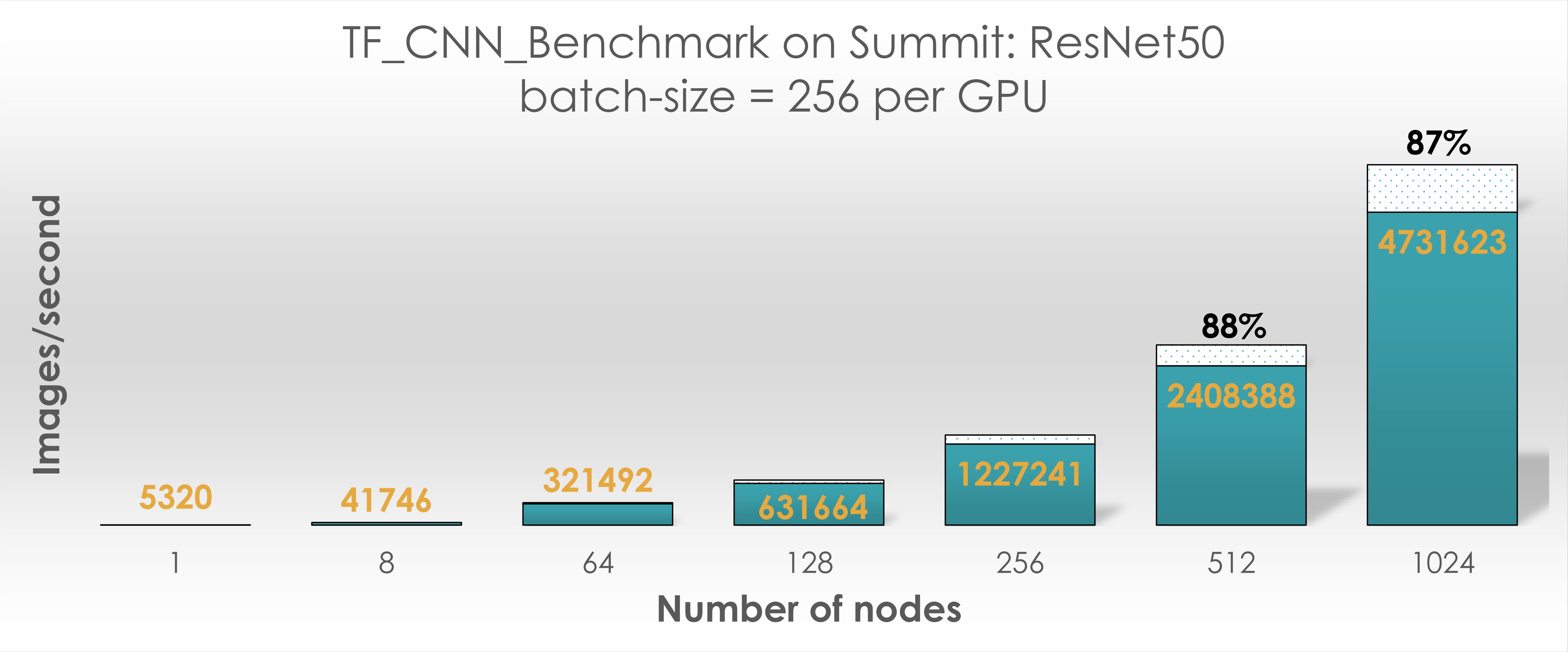


- Bit-Allreduce and Grouping generates overall 8x improvement in parallel efficiency

OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

Scaling considerations: putting it together

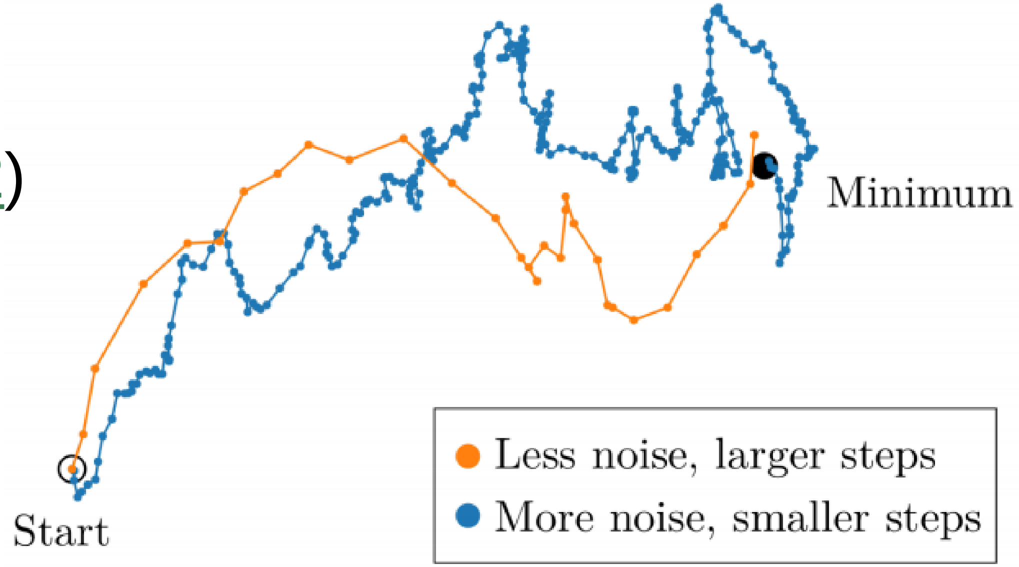
- ImageNet training

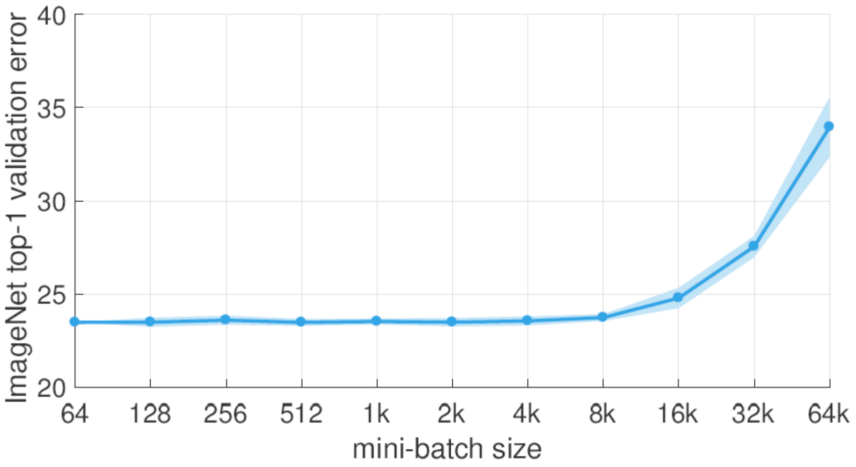


Scaling strategies: convergence considerations

- Large batch training: Optimal batch size ~ gradient noise scale ([arXiv: 1812.06162](https://arxiv.org/abs/1812.06162))
- Learning rate tuning: Layer-wise adaptive rate scaling (LARS) (arXiv:1711.04325)
- Scaling in time-to-solution is more challenging

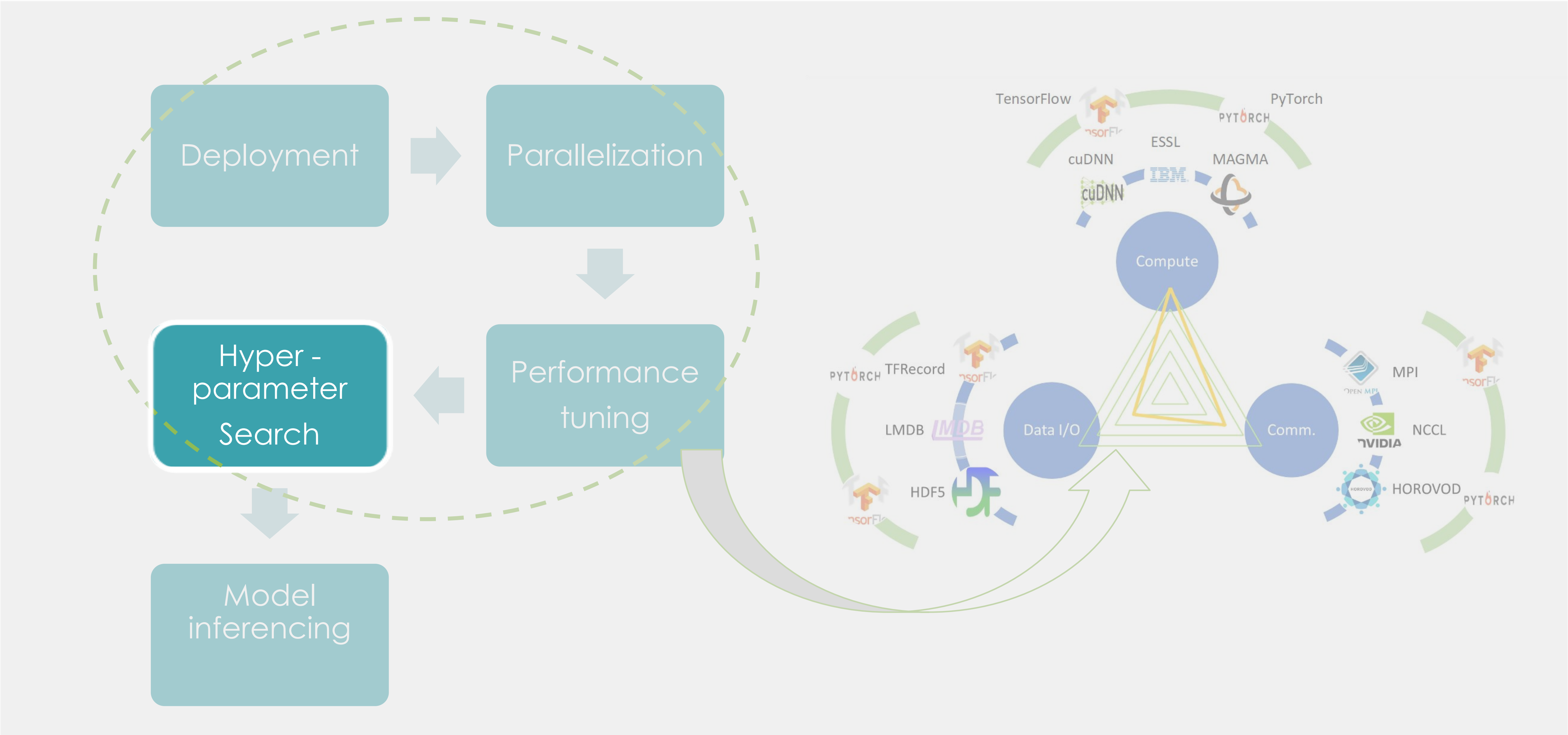
Nodes	Mini-batch size	Top-1 Val accuracy	Training time (min)
16	12288	0.750	27
32	12288	0.766	17
64	15360	0.763	12





(b) Empirical Accuracy (ResNet-50, figure adapted from [Goyal et al. 2017], lower is better)

Outline

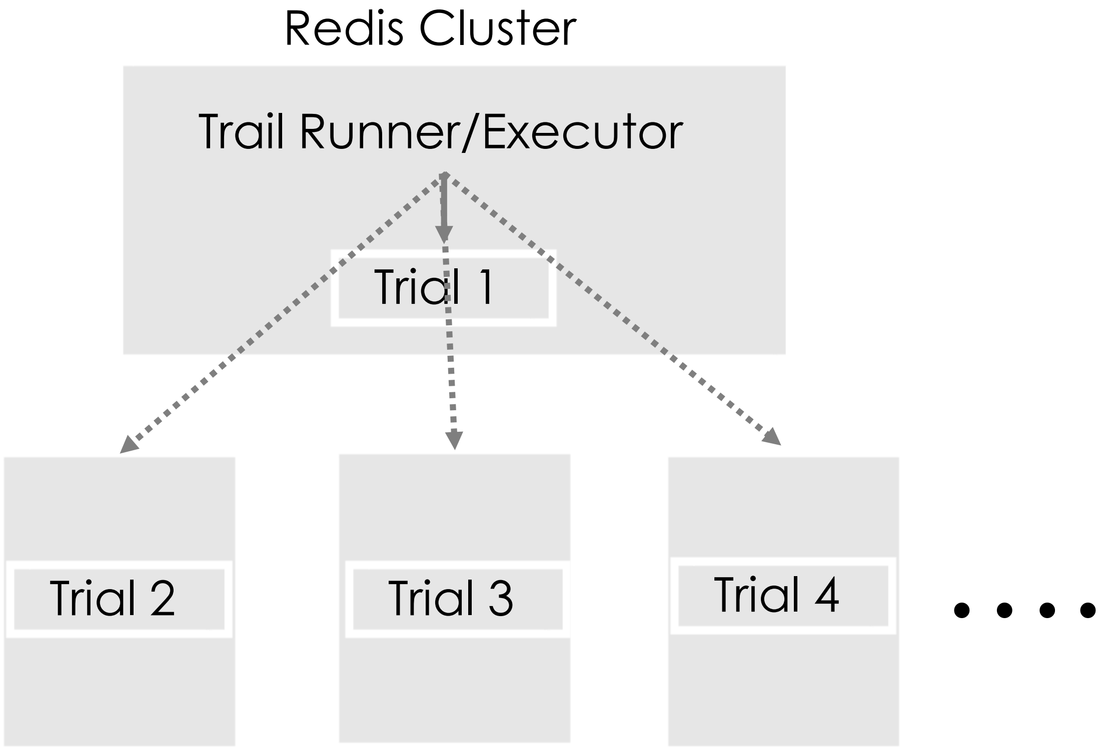


Deployment
Parallelization
Performance tuning
Hyper - parameter Search
Model inferencing
TensorFlow
PyTorch
ESSL
cuDNN
MAGMA
Compute
TFRecord
LMDB
HDF5
Data I/O
Comm.
MPI
NCCL
HOROVOD

Hyperparameter search with Ray Tune

https://ray.readthedocs.io/en/latest/tune.html

- Setup on Summit



- Scripts to start/stop Ray cluster

```
nodes=($(cat ${LSB_DJOB_HOSTFILE} | sort |
uniq | grep -v login | grep -v batch))
head=${nodes[0]}

ssh $head ray start --head --no-ui --redis-
port=6379 --temp-dir=$tmpdir --num-cpus=42 --
num-gpus=6

for worker in ${nodes[@]}; do
  ssh $worker ray start --redis-
address="$head:6379" --temp-dir=$tmpdir --
num-cpus=42 --num-gpus=6 &
  if [ $? -eq 0 ]; then
    echo "Ray worker started on $worker"
  fi
done
wait
```

Hyperparameter search with Ray Tune

- Using ray.tune.Trainable class

```
class Cifar10Model(Trainable):
  def _setup(self, config):
    model = self._build_model(depth=self.config["depth"])
    opt = tf.keras.optimizers.Adam(lr=self.config["lr"],
                                   decay=self.config["decay"])
  def _train(self):
    self.model.fit_generator(generator=gen, steps_per_epoch=self.config["batch_size"],
                             epochs=self.config["epochs"])
```

- Run experiments

```
ray.init(redis_address=args.redis_address)
pbt = PopulationBasedTraining(perturbation_interval=10,...)
run_experiments({"pbt_cifar10": train_spec}, scheduler=pbt)
```

https://code.ornl.gov/olcf-analytics/summit/distributed-deep-learning-examples/tree/master/examples/ray

Hyperparameter search with Ray Tune

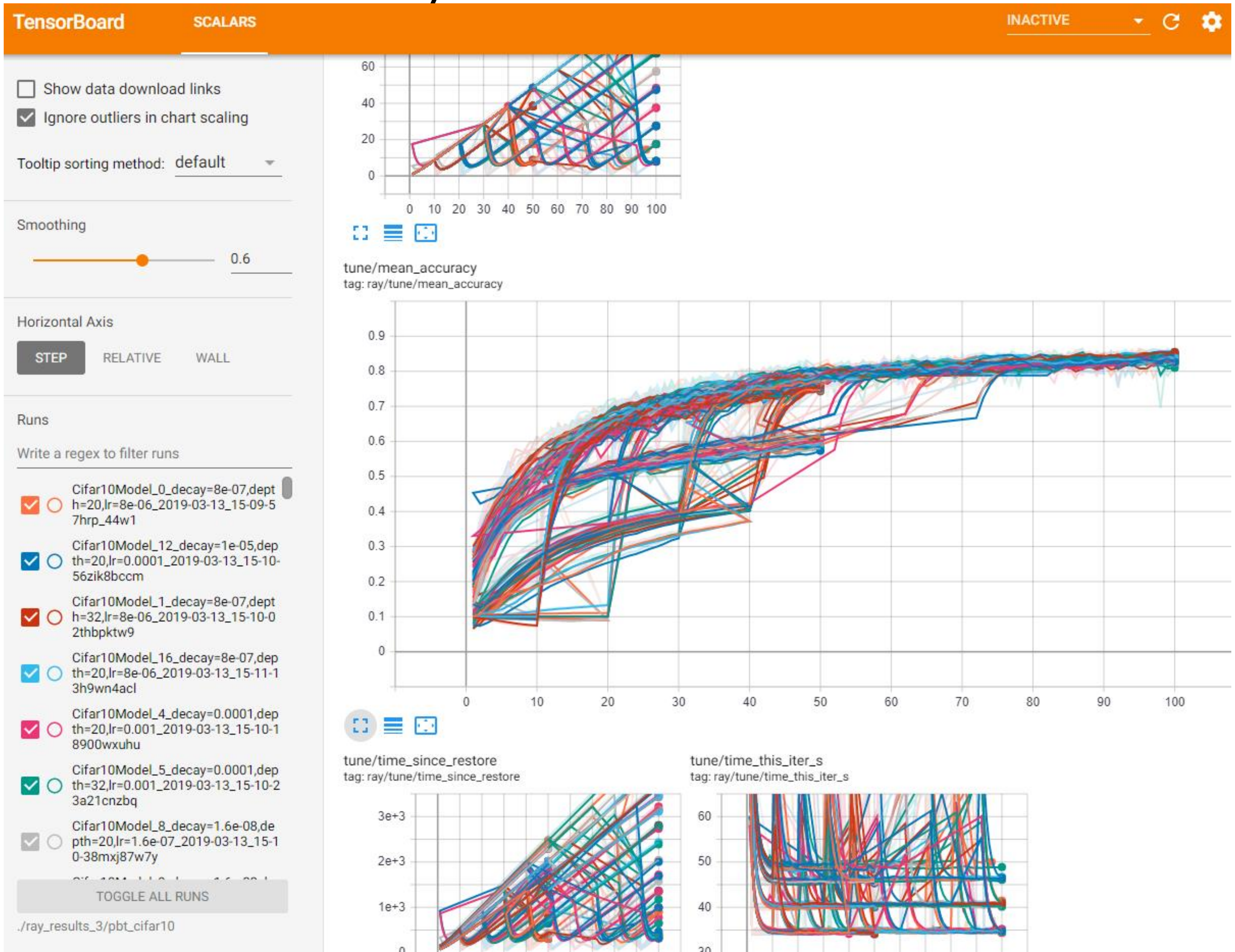
- Run experiments

```
train_spec = {
    "run": Cifar10Model,
    "resources_per_trial": {
        "cpu": 42,
        "gpu": 6
    },
    "stop": {
        "mean_accuracy": 0.90,
        "training_iteration": 50,
    },
    "config": {
        "epochs": 10,
        "batch_size": 64*6,
        "lr": grid_search([10**-3, 10**-4]),
        "decay": sample_from(lambda spec:
                                spec.config.lr / 10.0),
        "depth": grid_search([20,32,44,50]),
    },
}
```

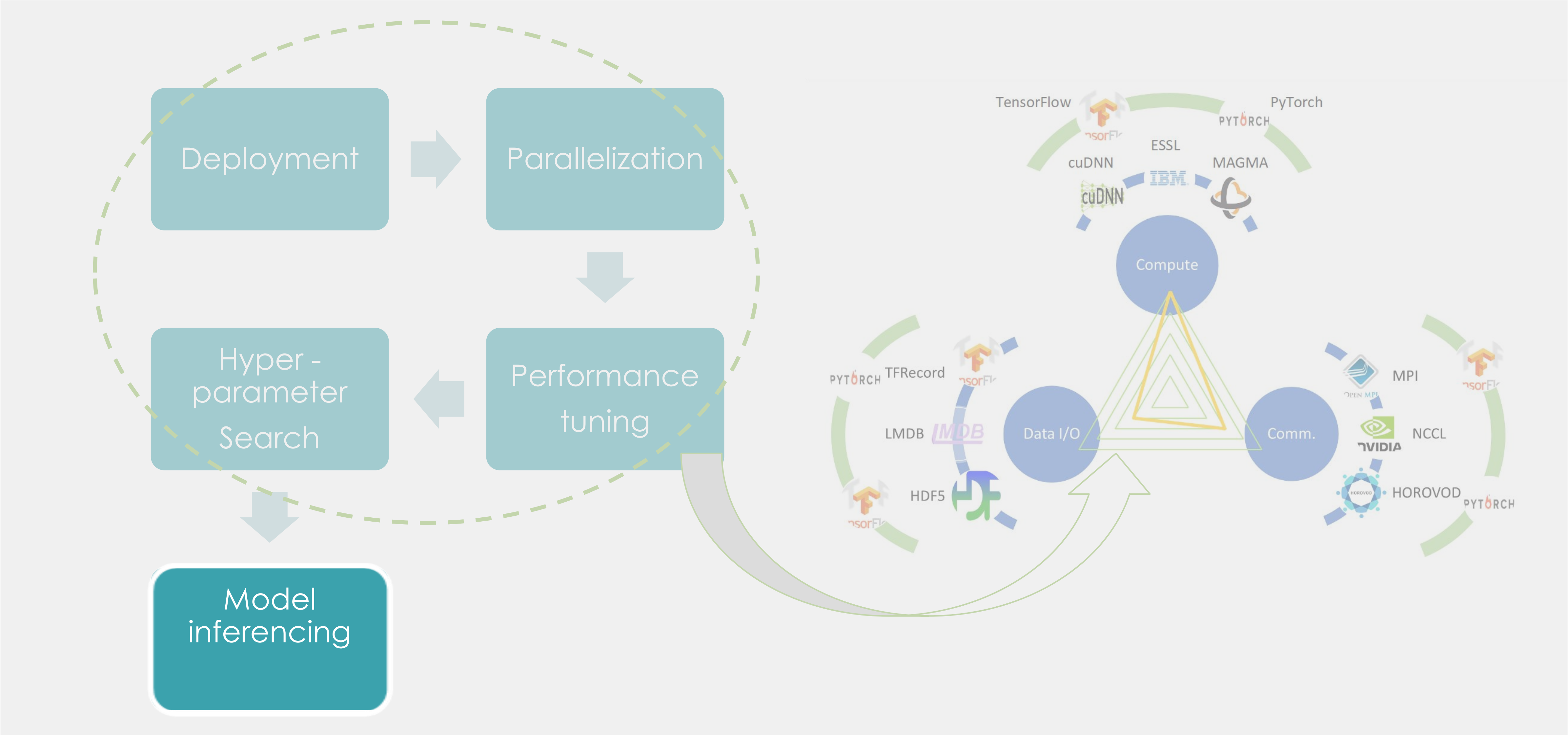
Main tuning parameters

Hyperparameter search with Ray Tune

- Vis with TensorBoard
  - TensorFlow
  - PyTorch
- Population based training example

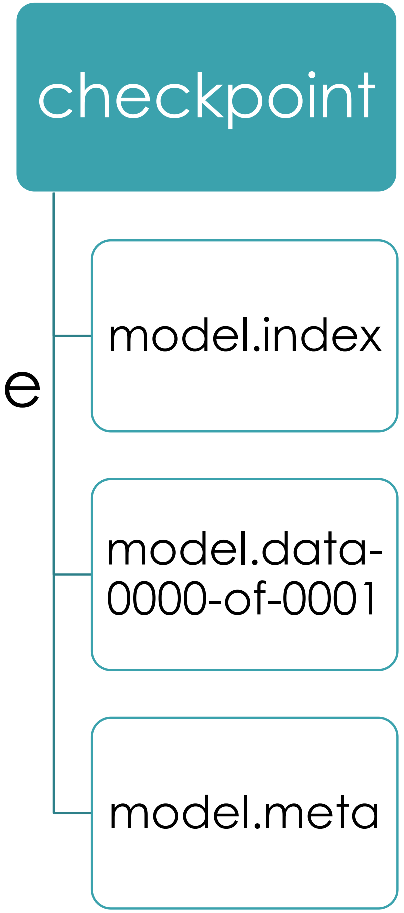


Outline


Deployment
Parallelization
Performance tuning
Hyper - parameter Search
Model inferencing
TensorFlow
PyTorch
ESSL
cuDNN
MAGMA
Compute
TFRecord
LMDB
HDF5
Data I/O
MPI
NCCL
HOROVOD
Comm.

Use TensorFlow C++ binding for Inferencing

- Train offline with Python interface
- Save the model checkpoint
  – Optimize the model
  tensorflow.python.tools.optimize_for_inference
- Deploy the model to simulation code with C++ interface
- Example use case: surrogate modeling



Binary files contain weights, gradients, and other variables

Protocol buffer file saves the TensorFlow graph

Use TensorFlow C++ binding for Inferencing

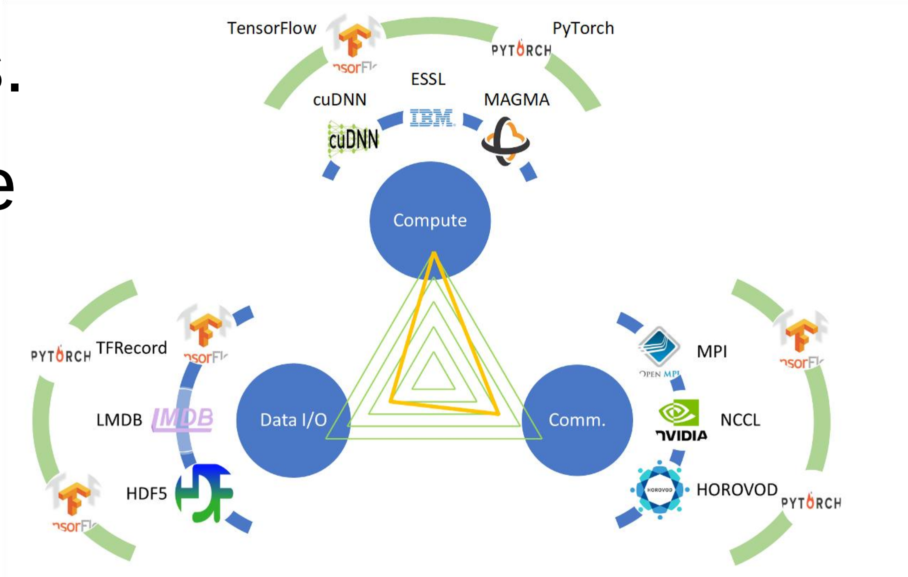
```
// Create TF session
tensorflow::Session sess; tensorflow::SessionOptions options
options.config.mutable_gpu_options()->set_visible_device_list(std::to_string(local_rank))
tensorflow::NewSession(options, sess)

// Load TF graph
tensorflow::MetaGraphDef graph_def;
ReadBinaryProto(tensorflow::Env::Default(), graph_file, &graph_def)
sess->Create(graph_def.graph_def())

// Load TF checkpoint
tensorflow::Tensor checkpointPathTensor(tensorflow::DT_STRING, tensorflow::TensorShape());
checkpointPathTensor.scalar<std::string>()() = checkpoint_file;
std::vector<std::pair<std::string, tensorflow::Tensor>> feed_dict = {
        {graph_def.saver_def().filename_tensor_name(), checkpointPathTensor}};

sess->Run(feed_dict, {}, {graph_def.saver_def().restore_op_name()}, nullptr);
```

Compile with TensorFlow C++ binding

```
module load ibm-wml-ce gcc/7.4.0

tf_include=$(python -c "import tensorflow as tf; import
sys; print (tf.sysconfig.get_include())")
tf_lib=$(python -c "import tensorflow as tf; import sys;
print (tf.sysconfig.get_lib())")

export CPATH=$tf_include:$CPATH
export LIBRARY_PATH=$tf_lib:$LIBRARY_PATH

mpic++ inference.cpp -ltensorflow_cc -ltensorflow_framework
-std=gnu++11
```

Conclusion

- Summit is ideal for deep learning applications.
- ImageNet training with ResNet50 can achieve 87% scaling efficiency up to 1024 nodes on Summit.
- Convergence issue of large-batch training may require hybrid-parallel to utilize full Summit parallelism.



Thank You!

For more information:
https://code.ornl.gov/olcf-analytics/summit/distributed-deep-learning-examples

J. Yin *et al*., "Strategies to Deploy and Scale Deep Learning on the Summit Supercomputer," *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, Denver, CO, USA, 2019, pp. 84-94.

Other resources

- TensorFlow data best practices: https://www.tensorflow.org/guide/data_performance
- TensorFlow distributed training: https://www.tensorflow.org/guide/distributed_training
- PyTorch data parallel: https://pytorch.org/tutorials/intermediate/ddp_tutorial.html
- Horovod examples: https://github.com/horovod/horovod/tree/master/examples
- Ray Tune documentation: https://ray.readthedocs.io/en/latest/tune.html