Deep Learning at Scale

Venkat Vishwanath
Data Sciences Group

venkat@anl.gov

November 7, 2018

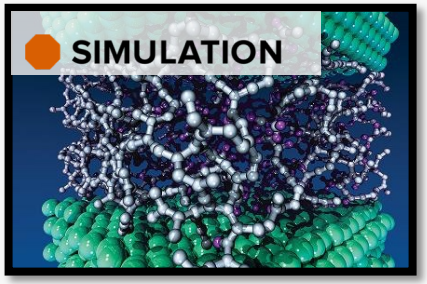
With some material from the 2018 ALCF Simulation, Data and Learning workshop
https://www.alcf.anl.gov/workshops/sdl-workshop-oct2018

MPI Tutorials
https://www.mcs.anl.gov/research/projects/mpi/tutorial/

Argonne Leadership Computing Facility

The Argonne Leadership Computing Facility provides world-class computing resources to the scientific community.

- Users pursue scientific challenges
- Resources fully dedicated to open science
- In-house experts to help maximize results



ALCF offers different pipelines based on your computational readiness. Apply to the allocation program that fits your needs.

https://www.alcf.anl.gov

Theta Intel/Cray

4,392 nodes
281,088 cores
69 TiB MCDRAM
824 TiB DDR4
549 TB SSD
Peak flop rate: 11.69 PF

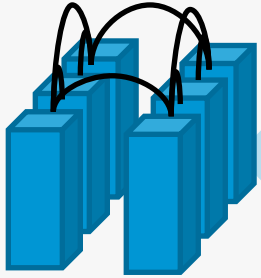
Mira IBM BG/Q
49,152 nodes
786,432 cores
786 TB RAM
Peak flop rate: 10 PF

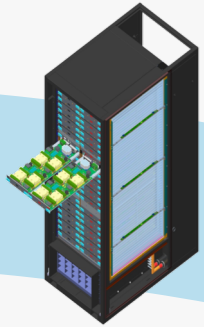
Argonne NATIONAL LABORATORY

Distributed memory architectures and a quick introduction on how to program them

Theta - System Overview

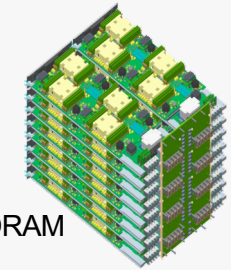
https://www.alcf.anl.gov/theta



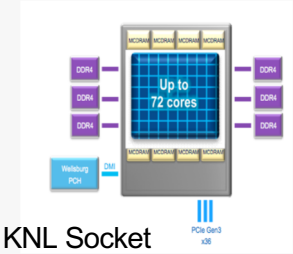


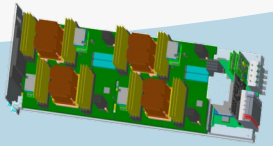
Cabinet: 3 Chassis, 75kW liquid/air cooled
510.72 TF 3TB MCDRAM, 36TB DRAM

System: 24 Cabinets
4392 Nodes, 1152 Switches
Dual-plane, 12 groups, Dragonfly 12.1 TB/s Bi-Sec
11.7 PF Peak
70 TB MCDRAM, 843 TB DRAM



Chassis: 16 Blades, 16 Cards
64 Nodes, 16 Switches
170.24 TF 1TB MCDRAM, 12TB DRAM



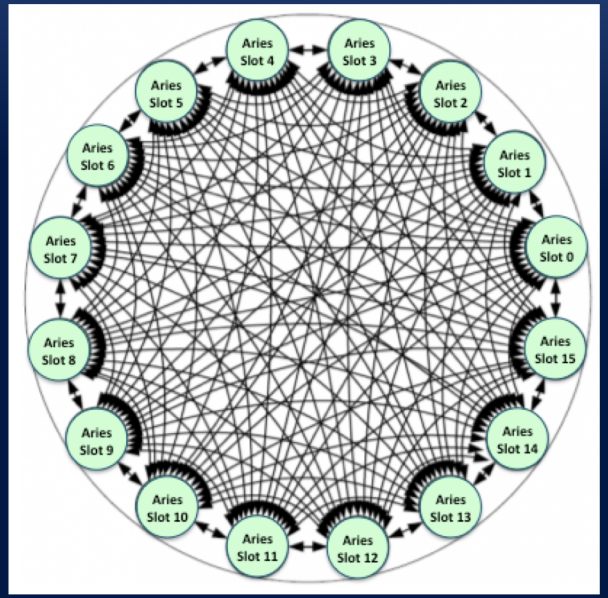


Compute Blade:
4 Nodes/Blade + Aries switch
10.64 TF 64GB MCDRAM
768GB DRAM

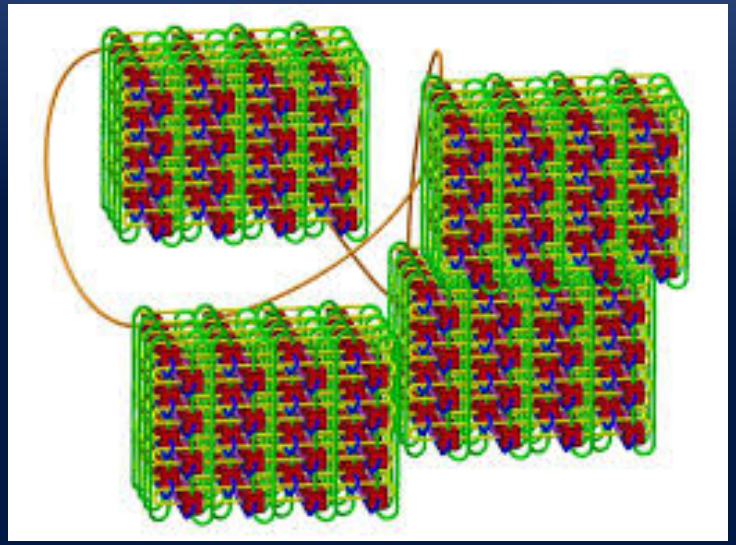
Sonexion Storage
4 Cabinets
Lustre file system
10 PB usable
210 GB/s

Node: KNL Socket
192 GB DDR4 (6 channels) **2.66 TF** 16GB MCDRAM
128 GB SSD

Supercomputer Interconnects



Dragonfly topology



5D Torus topology

Argonne NATIONAL LABORATORY

OLCF SUMMIT Supercomputer

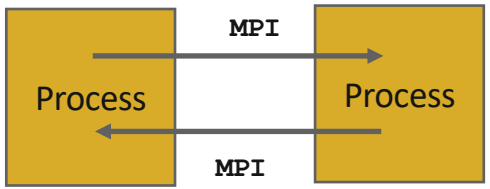
- Application Performance. 200 PF.
- Number of Nodes. 4,608.
- Node performance. 42 TF.
- Memory per Node. 512 GB DDR4 + 96 GB HBM2.
- NV memory per Node. 1600 GB.
- Total System Memory. >10 PB DDR4 + HBM2 + Non-volatile.
- Processors. 2 IBM POWER9™ 9,216 CPUs. 6 NVIDIA Volta™ 27,648 GPUs.
- File System. 250 PB, 2.5 TB/s, GPFS™

https://www.olcf.ornl.gov/summit/

Argonne NATIONAL LABORATORY

The Message-Passing Model

- A *process* is (traditionally) a program counter and address space.
- Processes may have multiple *threads* (program counters and associated stacks) sharing a single address space. MPI is for communication among processes, which have separate address spaces.
- Inter-process communication consists of
  - synchronization
  - movement of data from one process's address space to another's.



Source: Introduction to MPI, Argonne (06/06/2014)

Simple MPI Program

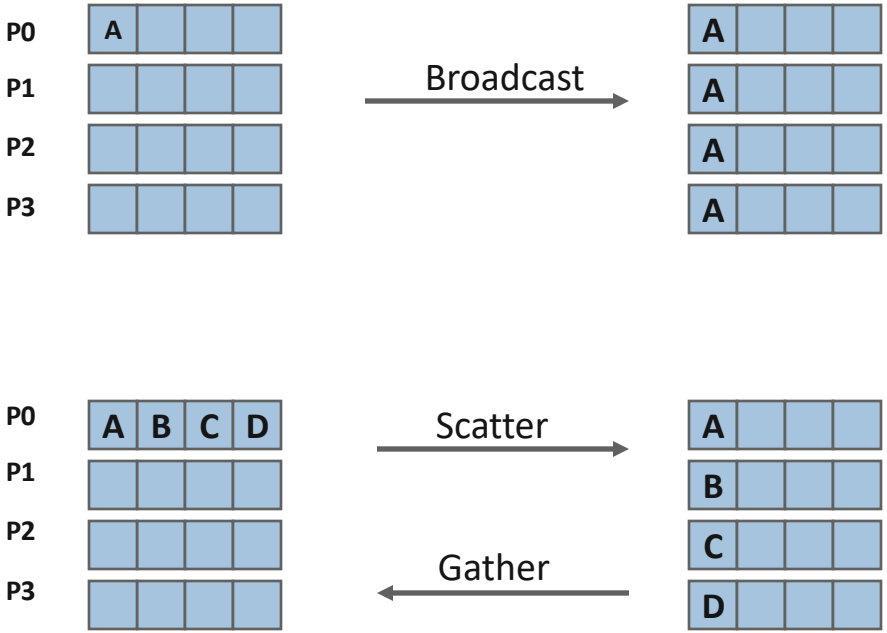
```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char ** argv)
{    int rank, size;
     MPI_Init(&argc, &argv);

     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
     MPI_Comm_size(MPI_COMM_WORLD, &size);
     printf("I am %d of %d\n", rank + 1, size);

     MPI_Finalize();
     return 0;
}
```

Basic requirements for an MPI program

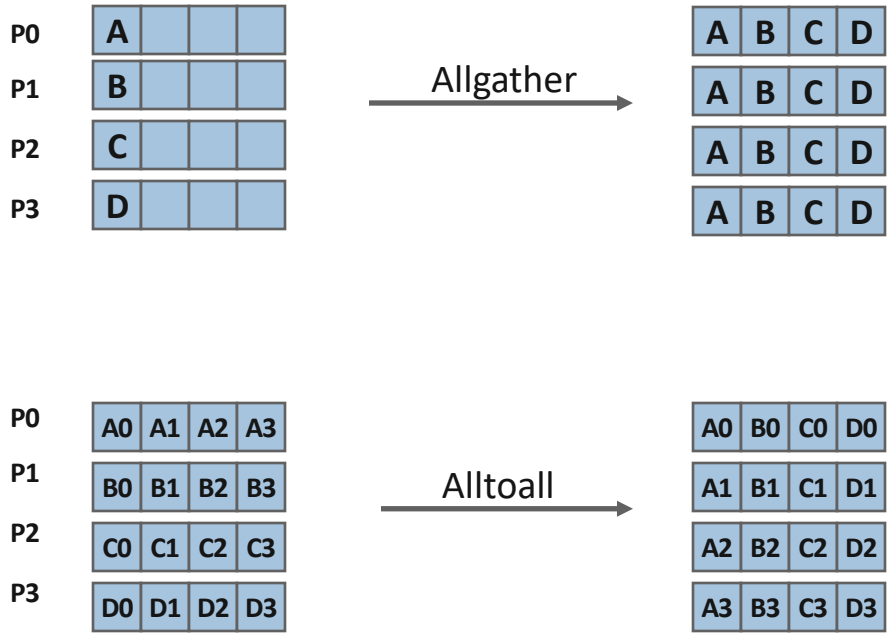
Simple MPI Communication Program

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char ** argv)
{
    int rank, data[100];

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0)
        MPI_Send(data, 100, MPI_INT, 1, 0, MPI_COMM_WORLD);
    else if (rank == 1)
        MPI_Recv(data, 100, MPI_INT, 0, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);

    MPI_Finalize();
    return 0;
}
```

Collective Data Movement



More Collective Data Movement

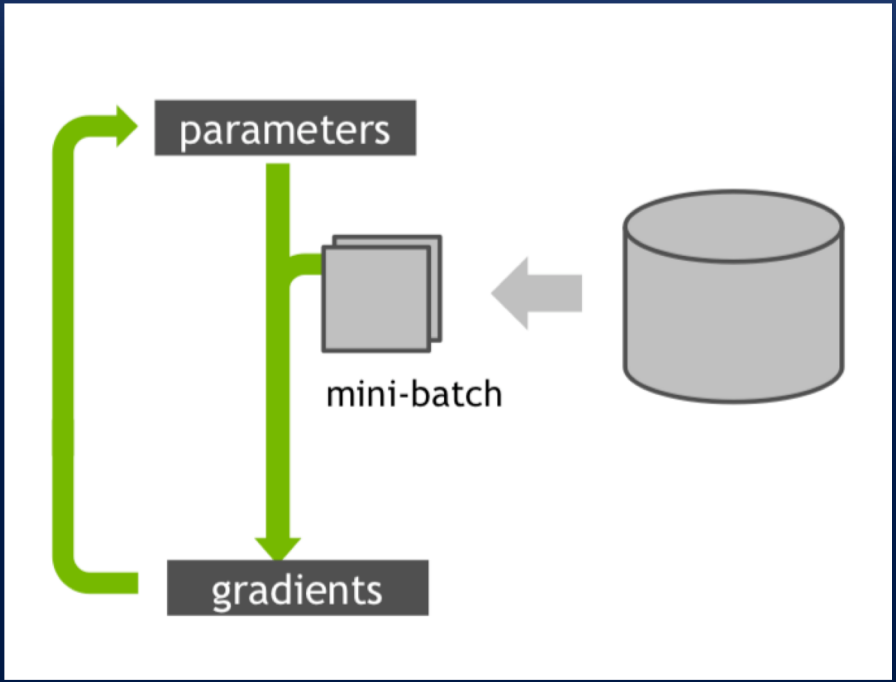


MPI4PY – Python and MPI

Demo and examples

Data Parallel Training

Mini-batch Training



Source: https://supercomputersfordl2017.github.io/Presentations/practical_scaling_techniques_v6.pdf

Why do we want to scale Deep Learning?

Deep Network Training

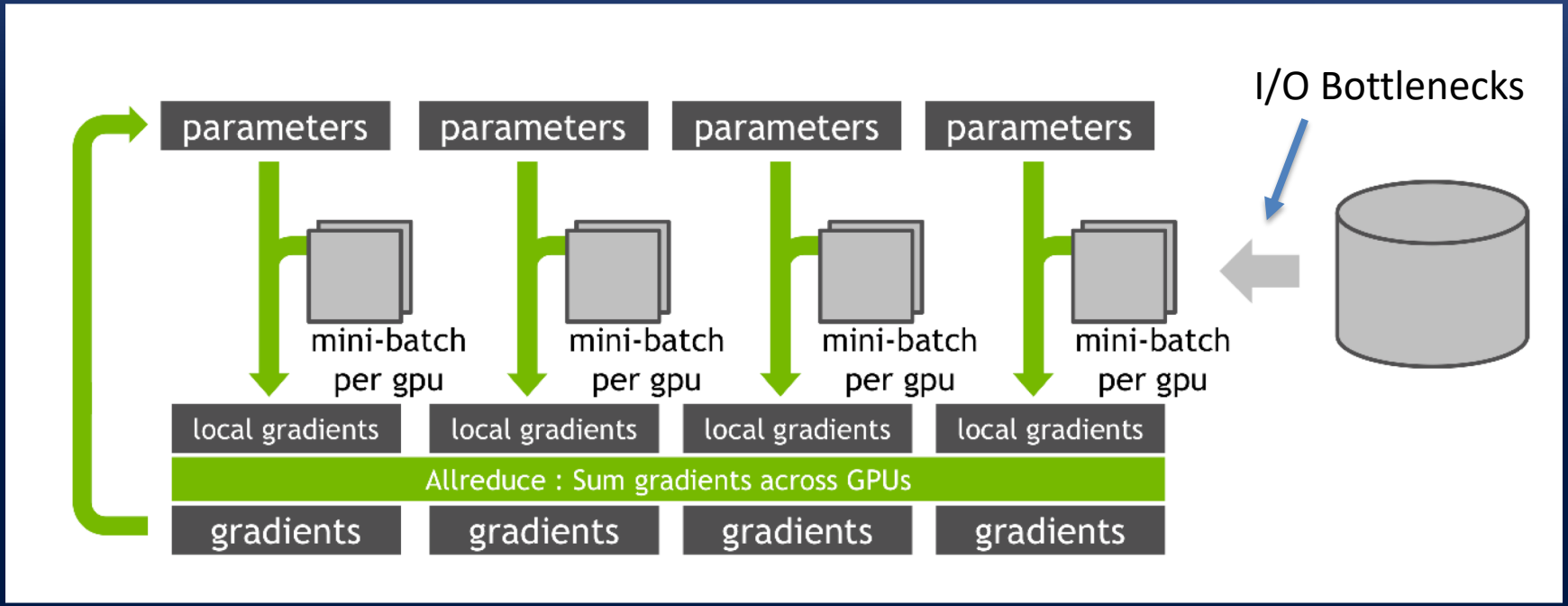
- We can strong scale training time-to-accuracy provided
- Number of workers (e.g., # nodes) << number of training examples
- Learning rate for particular batch size / scale is known
- Inference may have large scale computing needs, especially with UQ

Hyper-Parameter Optimization

- For problems and datasets where baseline accuracy is not known
  - learning rate schedule
  - momentum
  - batch size
- Evolve topologies if good architecture is unknown (common with novel datasets / mappings)
  - Layer types, width, number filters
  - Activation functions, drop-out rates

Source: Peter M.

Split the data across multiple nodes or sockets/GPUs per Node



Source: https://supercomputersfordl2017.github.io/Presentations/practical_scaling_techniques_v6.pdf

Data Parallel Synchronous SGD

- Data parallel training divides a global mini-batch of examples across processes
- Each process computes gradients from their local mini-batch
- Average gradients across processes
- All processes update their local model with averaged gradients (all processes have the same model)

Algorithm 1 Sync-SGD algorithm

for $0 \leq step < max\_steps$ **do**

$G_{local} \leftarrow$ COMPUTE_GRADIENTS(mini batch)

$G_{global} \leftarrow 1/N_{ranks} \times$ ALLREDUCE(G_{local})

APPLY_GRADIENTS(G_{global})

end for

Compute intensive

Communication intensive

Typically not much compute

- Not shown is the I/O activity of reading training samples (and possible augmentation)

Source: Peter M.

Horovod for Data Parallel Training

- https://github.com/uber/horovod
- De-facto data-parallel training on clusters and HPC systems
- Integrated with Tensorflow, Keras, PyTorch, PySpark, etc.
- Scaled to the largest supercomputers for deep learning
- For nodes with multiple GPUs, recommend using 1 mpi rank per GPU

Tensorflow with HOROVOD

```
import tensorflow as tf
import horovod.tensorflow as hvd
layers = tf.contrib.layers
learn = tf.contrib.learn
def main():
   # Horovod: initialize Horovod.
   hvd.init()
   # Download and load MNIST dataset.
   mnist = learn.datasets.mnist.read_data_sets('MNIST-data-%d' % hvd.rank())
   # Horovod: adjust learning rate based on number of GPUs.
   opt = tf.train.RMSPropOptimizer(0.001 * hvd.size())
   # Horovod: add Horovod Distributed Optimizer
   opt = hvd.DistributedOptimizer(opt)
   hooks = [
        hvd.BroadcastGlobalVariablesHook(0),
        tf.train.StopAtStepHook(last_step=20000 // hvd.size()),
        tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss},
                                   every_n_iter=10),
    ]
   checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None
   with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                          hooks=hooks,
                                          config=config) as mon_sess
```

https://github.com/uber/horovod/blob/master/examples/tensorflow_mnist.py

PyTorch with HOROVOD

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                    transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

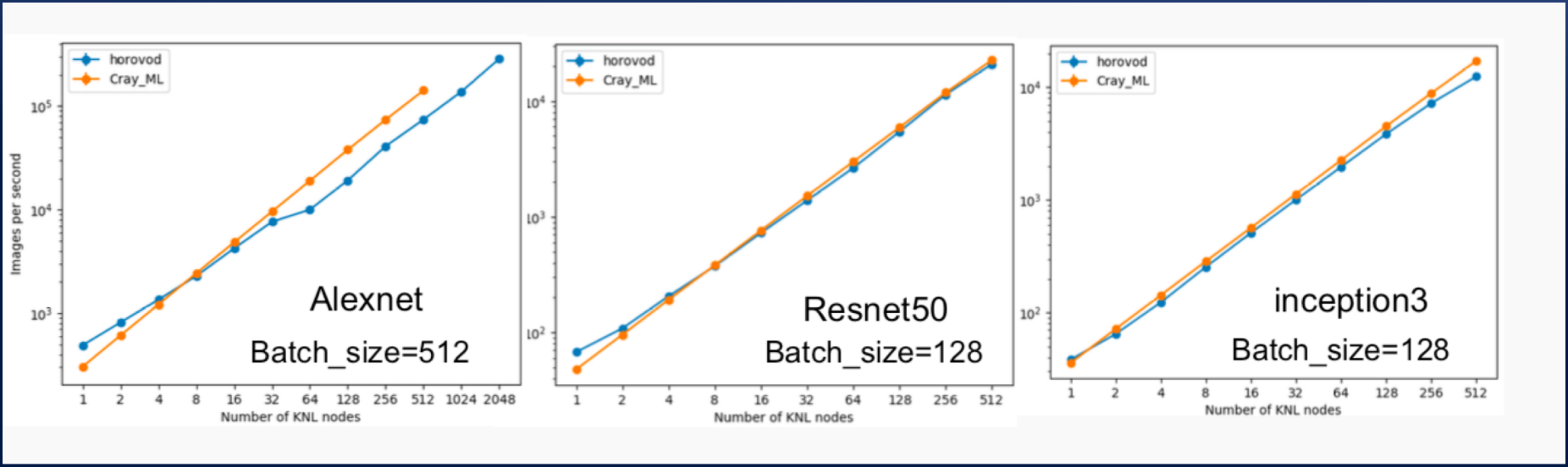
https://github.com/uber/horovod/blob/master/examples/pytorch_mnist.py

Keras with HOROVOD

```
import keras
import tensorflow as tf
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]
# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))
model.fit(x_train, y_train, batch_size=batch_size,
          callbacks=callbacks,
          epochs=epochs,
          verbose=1, validation_data=(x_test, y_test))
```

https://github.com/uber/horovod/blob/master/examples/keras_mnist.py

Data Parallel Training



Comparison between Cray ML plugin and Horovod scaling to half the Theta supercomputer (2048 nodes)

Argonne NATIONAL LABORATORY

Data Parallel Training Challenges

Increasing workers increases the global batch size

- This reduces the number of updates to the model (iterations) per
- epoch (full pass through dataset)
- Can require more iterations to converge to same validation accuracy for models trained at smaller batch sizes

Large-batch (LB) training can have different convergence properties than Small-batch (SB) training

- LB training can lead to models which fail to generalize to validation datasets
- LB training error can look similar to SB training error, but validation error fails to improve

Some Data Parallel Training Scaling Notes

- **Step 1) Start with common initial learning rate for selected optimizer (from Keras documentation)**
  - Adam -> 0.001
  - RMSProp -> 0.001
  - SGD -> 0.01
  - Adagrad -> 0.01
  - Adadelta -> 1.0
- **Step 2) Multiply learning rate by N or Sqrt(N)**
  - N is the number of parallel processes
  - Discussed in further detail on next slide
- **Step 3) Decay learning rate during training (e.g., exponential decay)**
  - Setup a learning rate schedule using your initial learning rate as the starting state
  - Learning rate typically lowered periodically or continuously
  - Helps improve final accuracy
  - Likely very important to reduce learning rate over time when initial learning rate scaled large
- **Step 4) Run it**
  - Train and observe loss or training accuracy, check validation accuracy
  - Adjust initial learning rate up if learning too slowly or down if model is not learning
  - Repeat steps as needed to improve convergence and accuracy

Learning Rate Scaling Rules

- **Sqrt Scaling Rule:**
  - When the local minibatch size is multiplied by $N_{workers}$, multiply the learning rate by $\sqrt{N_{workers}}$.

$$\eta_{init} = \eta_{init} * \sqrt{N_{workers}}$$

  - Error on the mean only improves as sqrt(N_workers)
- **Linear Scaling Rule:**
  - When the minibatch size is multiplied by N, multiply the learning rate by N.

$$\eta_{init} = \eta_{init} * N_{workers}$$

  - Naïve rule for scaling learning rate in distributed training but it works for some problems
  - More attractive (when it works) because it shouldn't require many additional iterations to reach same accuracy

ImageNet in 1 Hour

- *“the linear scaling rule breaks down when the network is changing rapidly, which commonly occurs in early stages of training. We find that this issue can be alleviated by a properly designed warmup, namely, a strategy of using less aggressive learning rates at the start of training.”*
- Design a **warm-up set of iterations** to reduce these errors. Once training settled on good path (5-15 epochs), transition to larger learning rate

https://research.fb.com/wp-content/uploads/2017/06/imagenet1kin1h5.pdf

LARS – Layer-Wise Adaptive Rate Scaling
Imagenet In ~30 mins

Table 2: AlexNet-BN: The norm of weights and gradients at 1st iteration.

Layer	conv1.b	conv1.w	conv2.b	conv2.w	conv3.b	conv3.w	conv4.b	conv4.w
$\|\|w\|\|$	1.86	0.098	5.546	0.16	9.40	0.196	8.15	0.196
$\|\|\nabla L(w)\|\|$	0.22	0.017	0.165	0.002	0.135	0.0015	0.109	0.0013
$\frac{\|\|w\|\|}{\|\|\nabla L(w)\|\|}$	8.48	**5.76**	33.6	83.5	69.9	127	74.6	148
Layer	conv5.b	conv5.w	fc6.b	fc6.w	fc7.b	fc7.w	fc8.b	fc8.w
$\|\|w\|\|$	6.65	0.16	30.7	6.4	20.5	6.4	20.2	0.316
$\|\|\nabla L(w)\|\|$	0.09	0.0002	0.26	0.005	0.30	0.013	0.22	0.016
$\frac{\|\|w\|\|}{\|\|\nabla L(w)\|\|}$	73.6	69	117	**1345**	68	489	93	19

If LR is large comparing to the ratio for some layer, then training may becomes unstable. The LR "warm-up" attempts to overcome this difficulty by starting from small LR, which can be safely used for all layers, and then slowly increasing it until weights will grow up enough to use larger LRs.

Algorithm 1 SGD with LARS. Example with weight decay, momentum and polynomial LR decay.

Parameters: base LR γ_0, momentum m, weight decay β, LARS coefficient η, number of steps T
Init: $t = 0$, $v = 0$. Init weight w_0^l for each layer l
while $t < T$ for each layer l **do**
  $g_t^l \leftarrow \nabla L(w_t^l)$ (obtain a stochastic gradient for the current mini-batch)
  $\gamma_t \leftarrow \gamma_0 * \left(1 - \frac{t}{T}\right)^2$ (compute the global learning rate)
  $\lambda^l \leftarrow \frac{||w_t^l||}{||g_t^l|| + \beta ||w_t^l||}$ (compute the local LR λ^l)
  $v_{t+1}^l \leftarrow m v_t^l + \gamma_{t+1} * \lambda^l * (g_t^l + \beta w_t^l)$ (update the momentum)
  $w_{t+1}^l \leftarrow w_t^l - v_{t+1}^l$ (update the weights)
end while

The network training for SGD with LARS are summarized in the Algorithm 1. One can find more implementation details at https://github.com/borisgin/nvcaffe-0.16

The local LR strongly depends on the layer and batch size (see Figure. 2)

- **Key idea: Layer-wise learning rate instead of a single learning rate**
- **A great example of interpretability and understanding how the norm of weight and gradients change for various layers**
- **Demonstrated no loss of accuracy on ResNet50 with global batch size of 32K**

Large Batch Training of Convolutional Networks
https://arxiv.org/pdf/1708.03888.pdf and https://openreview.net/pdf?id=rJ4uaX2aW

Argonne NATIONAL LABORATORY

ImageNet in 15 Minutes

Use different optimizers for each phase, started with RMSProp first and then moved to SGD

Table 1: 90-epoch training time and single-crop validation accuracy of ResNet-50 for ImageNet reported by different teams.

Team	Hardware	Software	Minibatch size	Time	Accuracy
He *et al.* [5]	Tesla P100 × 8	Caffe	256	29 hr	75.3 %
Goyal *et al.* [4]	Tesla P100 × 256	Caffe2	8,192	1 hr	76.3 %
Codreanu *et al.* [3]	KNL 7250 × 720	Intel Caffe	11,520	62 min	75.0 %
You *et al.* [10]	Xeon 8160 × 1600	Intel Caffe	16,000	31 min	75.3 %
This work	Tesla P100 × 1024	Chainer	32,768	15 min	74.9 %

Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes
https://arxiv.org/pdf/1711.04325.pdf

Training ImagNet in 4 minutes

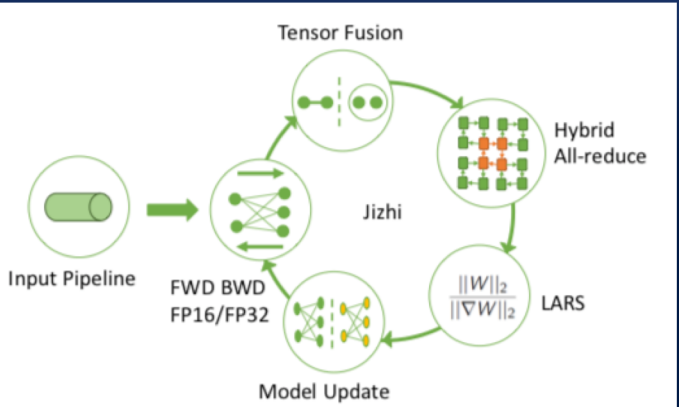


Figure 1: Jizhi Training System Overview

Figure 2: Mix-precision training with LARS

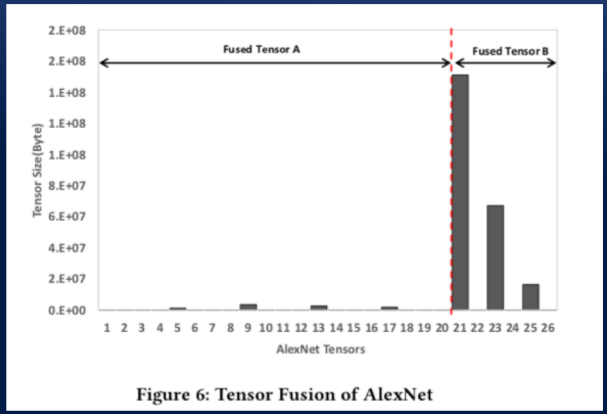


Figure 6: Tensor Fusion of AlexNet

Overall architecture

Optimization 1: Mixed precision training

Optimization 2: Tensor Fusion

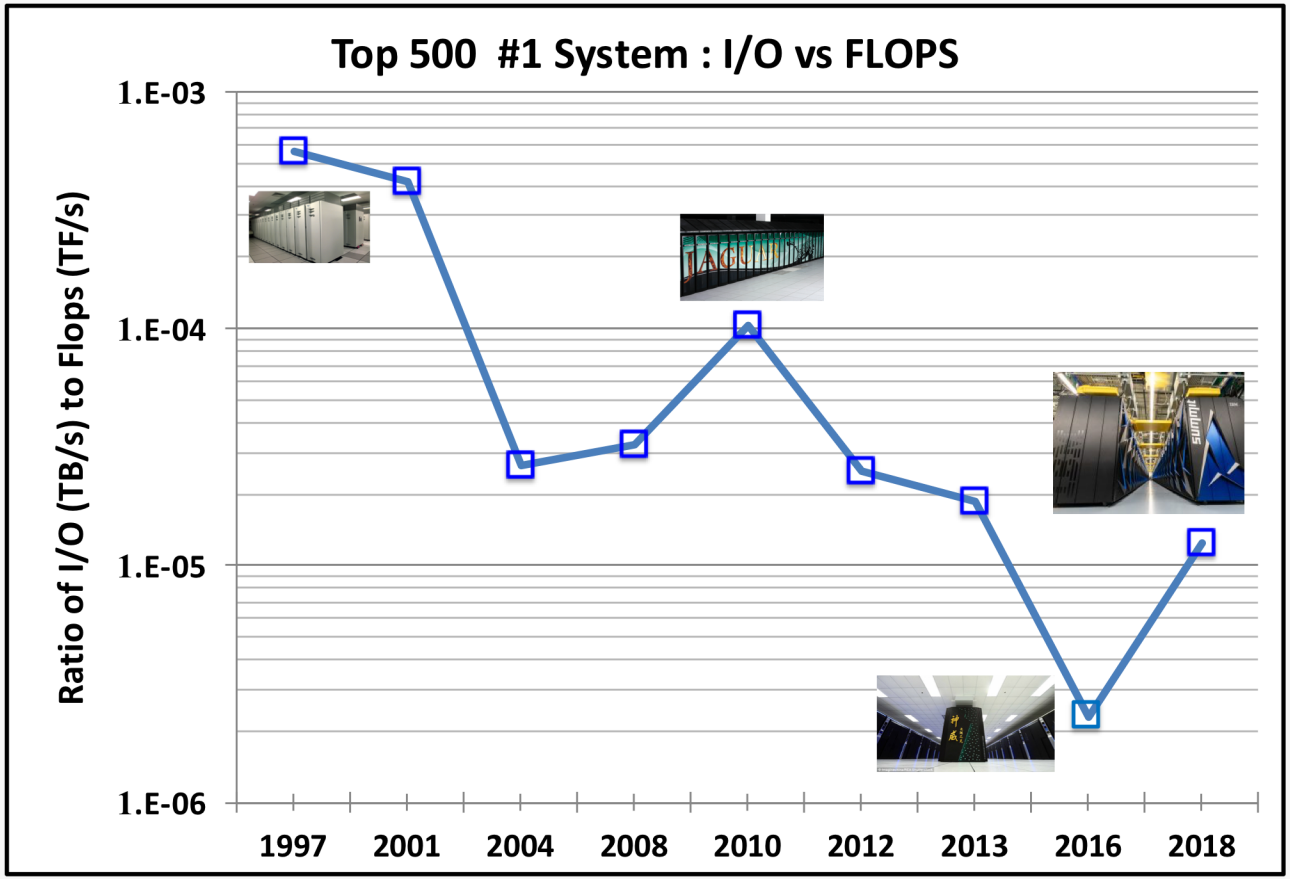
https://arxiv.org/pdf/1807.11205.pdf

Argonne NATIONAL LABORATORY

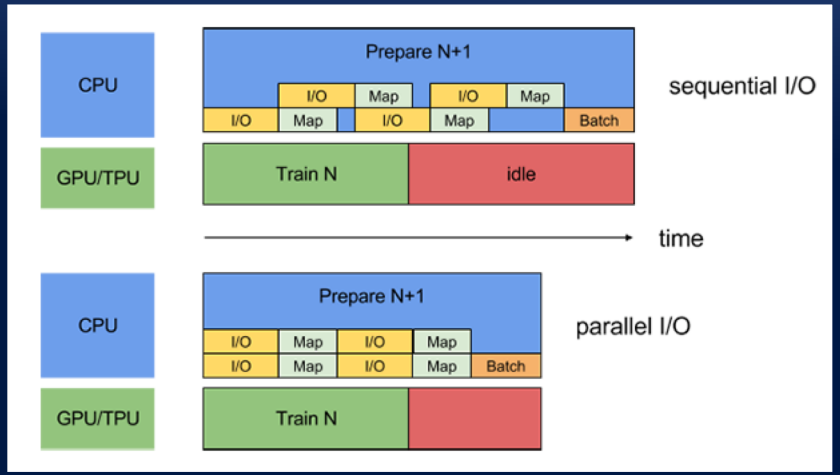
Other optimizations

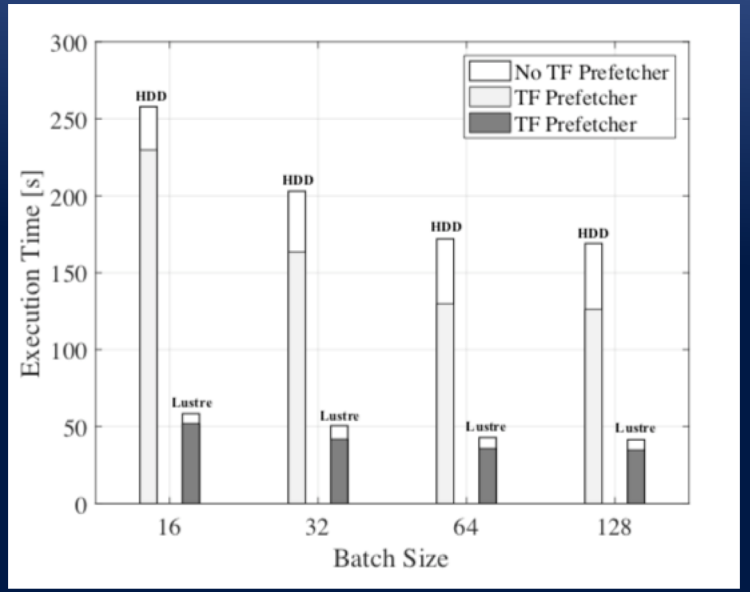
- Non-blocking collective-based updates for each layer
  - Overlaps computation for a layer ***n*** with communication for the lower layer ***n-1***
  - *Lagging* to overcome the top-most layer
- I/O and data staging

Historically: Compute has Outpaced I/O



Improving data reading and staging





https://www.tensorflow.org/guide/performance/datasets

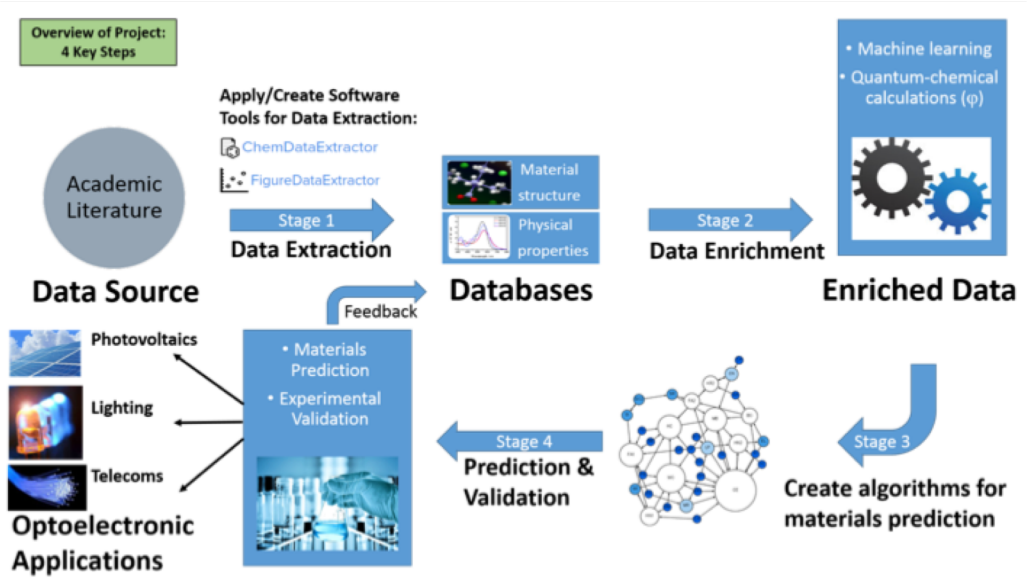
https://arxiv.org/pdf/1810.03035.pdf

Learning in Science Examples

Data-Driven Molecular Engineering of Solar-Powered Windows

PI: Jacqueline Cole, Cambridge University, Argonne National Laboratory. 117M core hours

- **Objective:** Better light-absorbing dye molecules
- **Impact:** Dye-sensitized solar cells (DSCs) are an alternative to organic metal cells with a desirable cost-efficiency tradeoff in the design of energy-efficient buildings. 40% of total energy usage in the USA comes from buildings.
- **Approach:** Extracting data from 300,000 publications and computing properties of 80,000 molecules require leadership computing resources .Uses quantum chemistry using TDDFT. Explored viable molecular descriptors to evaluate candidate molecules using ML

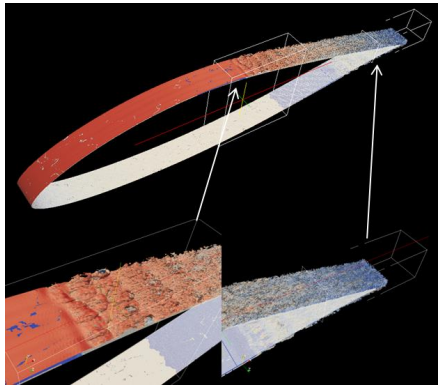


Discovered a new class of DSCs that are based on a weak intramolecular interaction between sulfur and carbon atoms.

J. M. Cole *et al*, *ACS Appl. Mater. Interfaces*, **2017**, *9* (31),25952

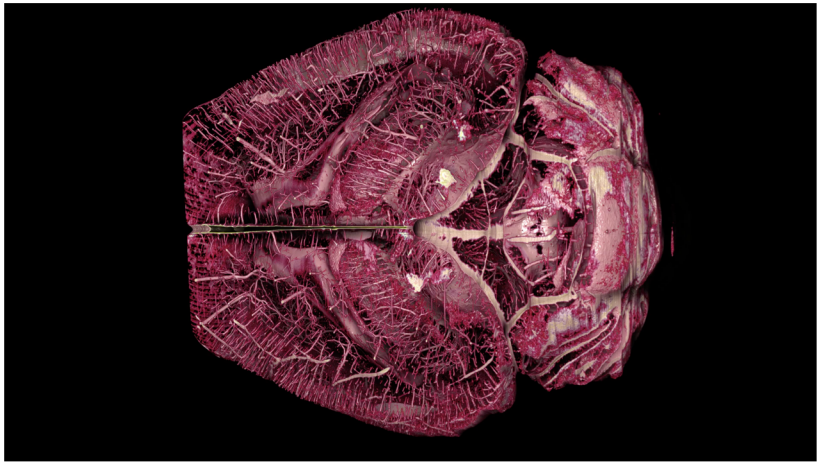
Argonne NATIONAL LABORATORY

DATA AND LEARNING PROJECTS

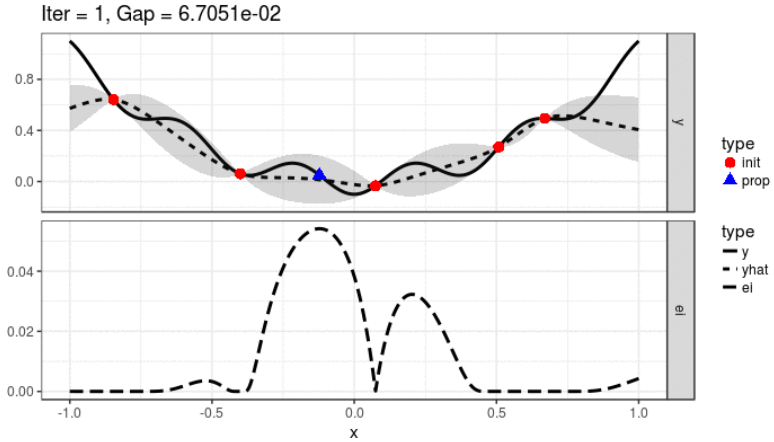


Leverage machine learning and large data sets generated by well resolved LES simulations to develop data driven turbulence models

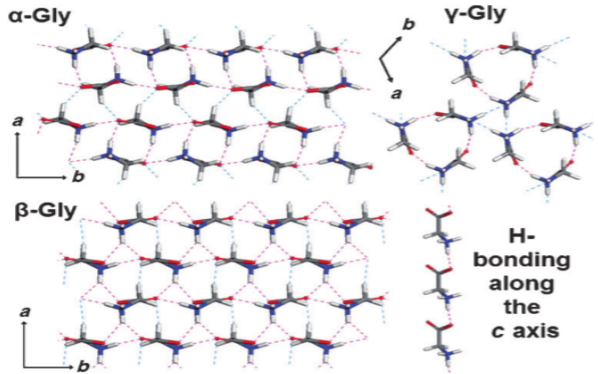
Engineering, GE Global Research



Neuroscience and Imaging, Argonne



DeepHyper, Scalable Hyperparameter Opt



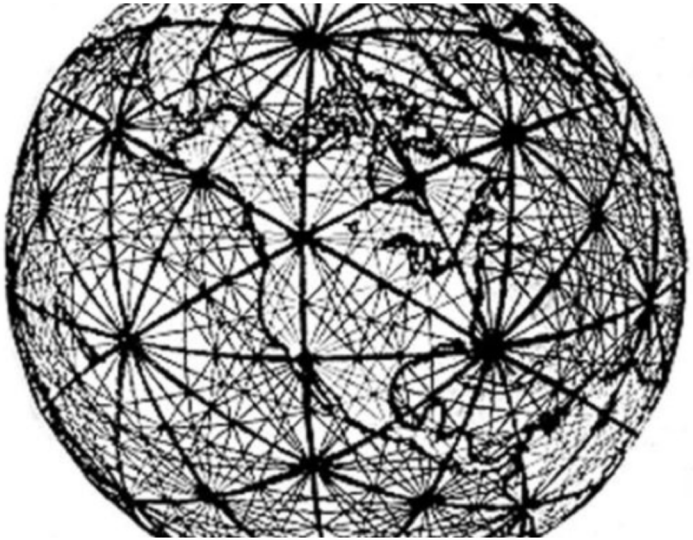
Many-body interactions prediction

Argonne NATIONAL LABORATORY

Gordon Bell 2018 finalists used Deep Learning

DEEP LEARNING JUST DIPPED INTO EXASCALE TERRITORY

October 5, 2018 Nicole Hemsoth

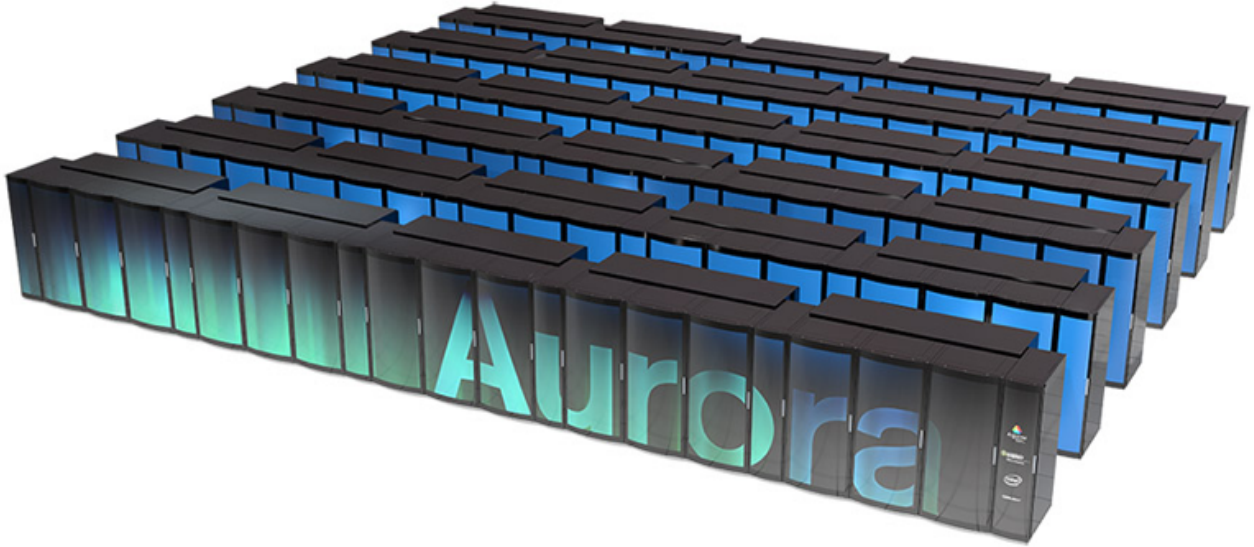


> 1 Exaops performance in using Deep learning for segmentation in climate science using the Volta Tensor cores

Scaled to ~27K V100 GPU on the Summit supercomputer

https://www.nextplatform.com/2018/10/05/deep-learning-just-dipped-into-exascale-territory/

Aurora 2021 (A21) The first US Exascale System



Architecture supports three ways of computing

- **Large-scale Simulation (PDEs, traditional HPC)**
- **Data Intensive Applications (scalable science pipelines)**
- **Deep Learning and Emerging Science AI (training and inferencing)**

Thank you!!

venkat@anl.gov