MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Follow us on

https://twitter.com/mvapich

HiDL
High-Performance Deep Learning

# High Performance Machine Learning, Deep Learning, and Data Science with MVAPICH2

## Tutorial at MUG '22

by


**Aamir Shafi**

The Ohio State University

shafi.16@osu.edu

https://cse.osu.edu/people/shafi.16

**Arpan Jain**

The Ohio State University

jain.575@osu.edu

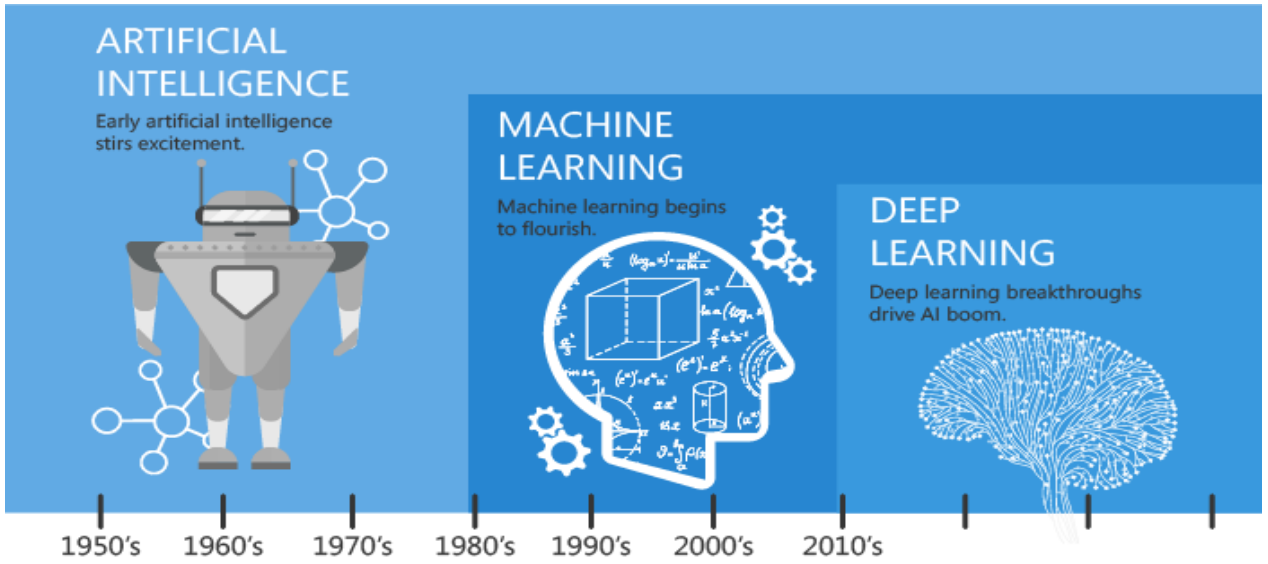
http://u.osu.edu/jain.575

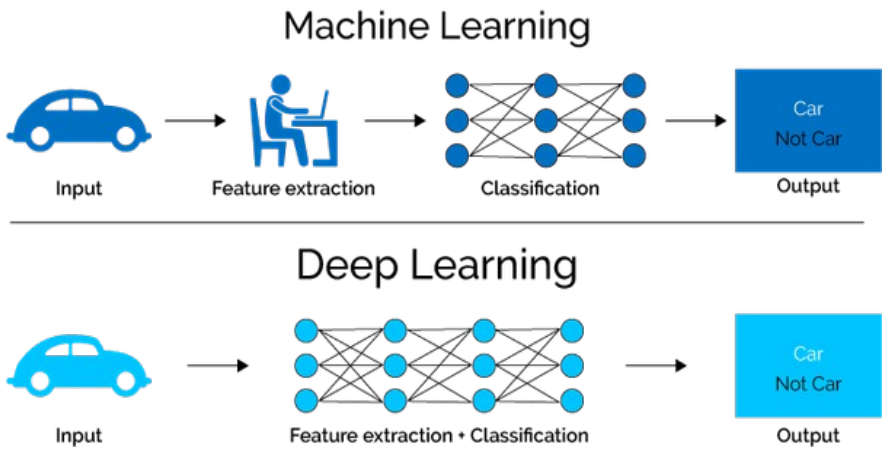
# Outline

- **Introduction**
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- Deep Learning
  - Deep Neural Networks
  - Distributed Deep Learning
  - DL Solutions
- Data Science
- Conclusion

# What is Machine Learning and Deep Learning?

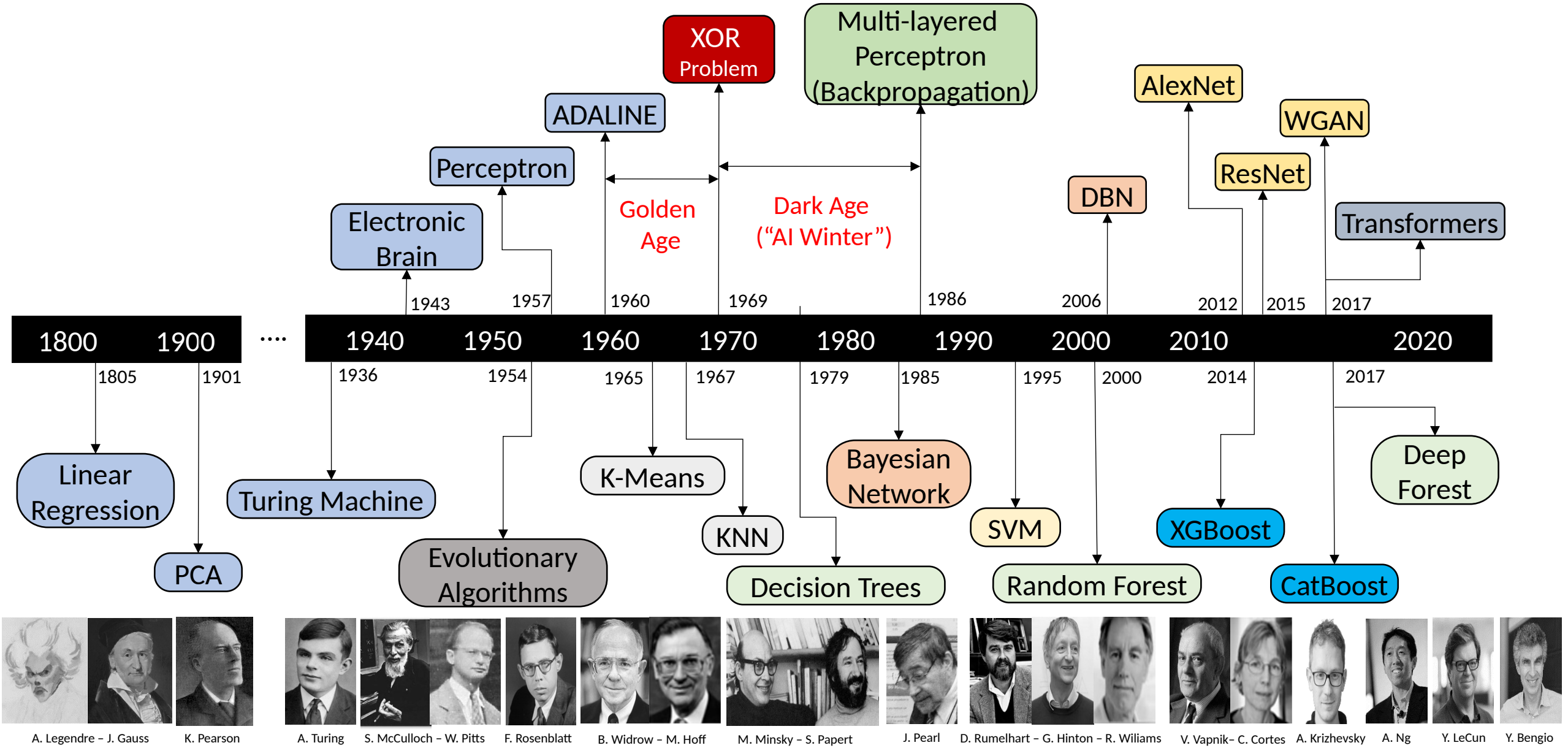
- Machine Learning (ML)
  - "the study of computer algorithms to improve automatically through experience and use of data"
- Deep Learning (DL) – a subset of ML
  - Uses Deep Neural Networks (DNNs)
  - **Perhaps, the most revolutionary subset!**
- Based on learning data representation
- DNN Examples: Convolutional Neural Networks, Recurrent Neural Networks, Hybrid Networks
- Data Scientist or Developer Perspective for using DNNs
  1. Identify DL as solution to a problem
  2. Determine Data Set
  3. Select Deep Learning Algorithm to Use
  4. Use a large data set to train an algorithm




**Courtesy:**
https://hackernoon.com/difference-between-artificial-intelligence-machine-learning-and-deep-learning-1pcv3zeg
, https://blog.dataiku.com/ai-vs.-machine-learning-vs.-deep-learning,

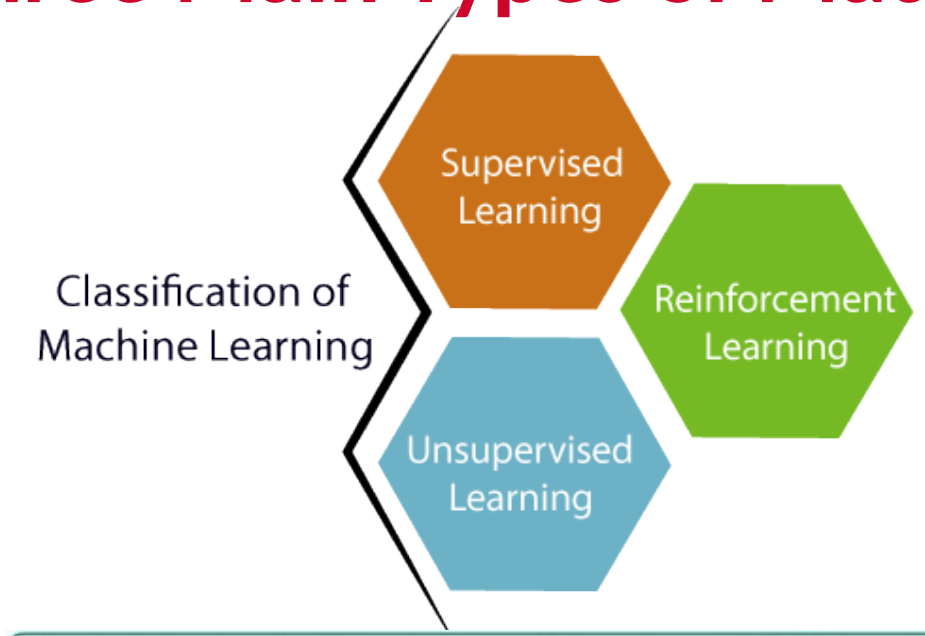
# History: Milestones in the Development of ML/DL

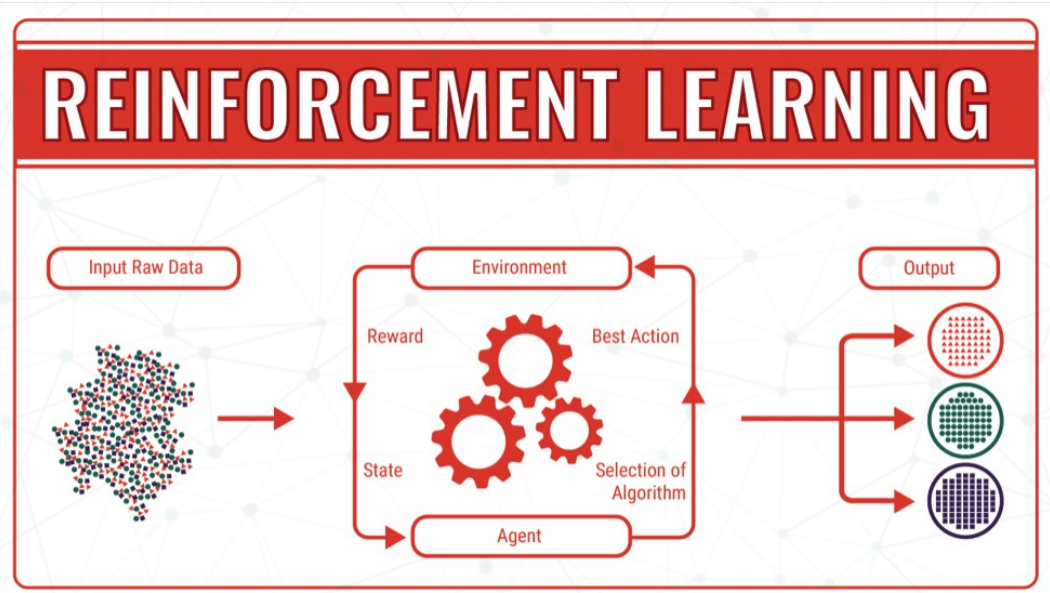


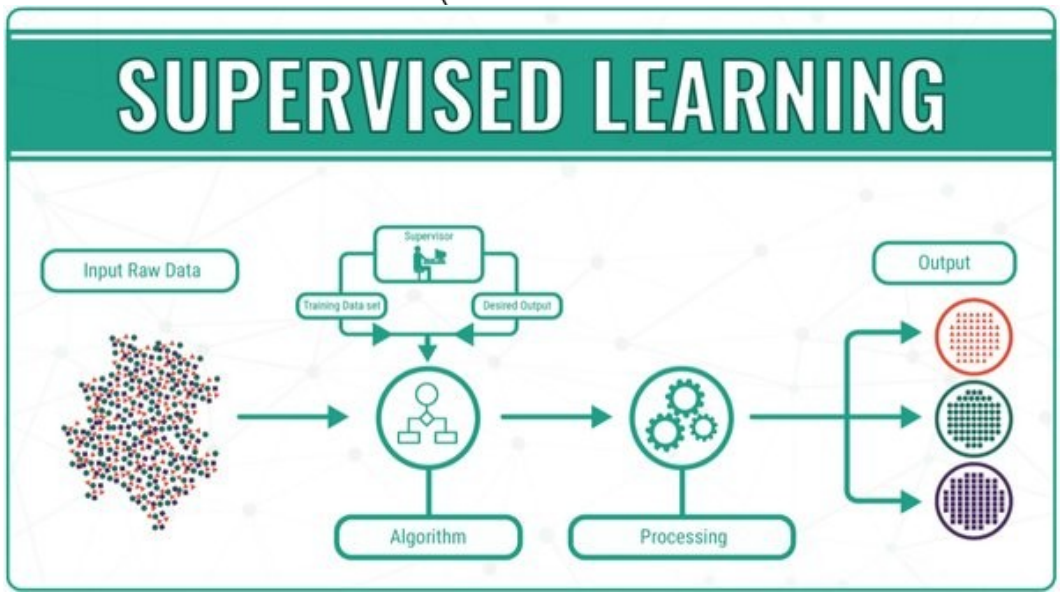
# Outline

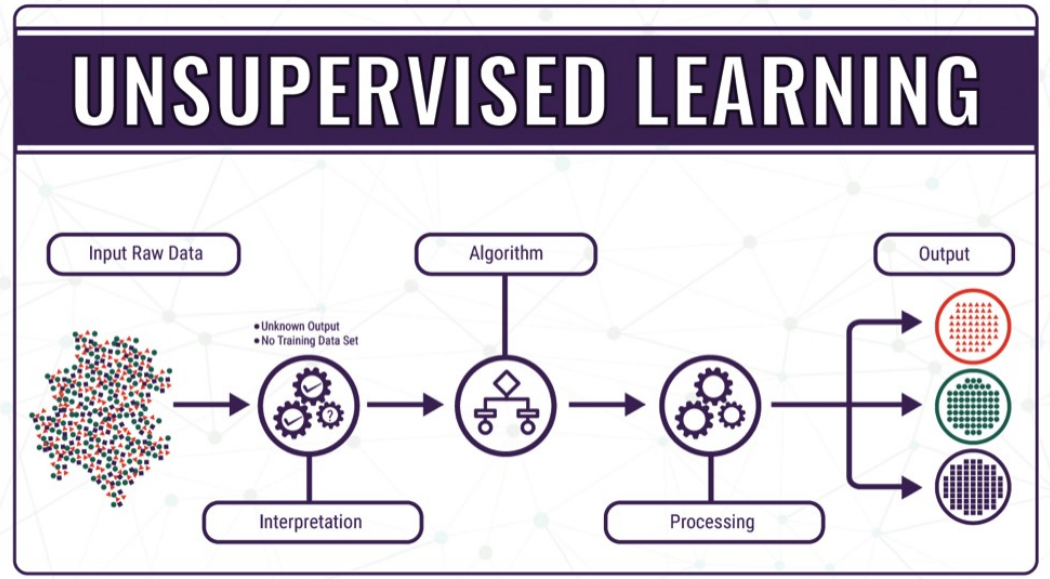
- Introduction
- **Machine Learning**
  - **Distributed K-Means**
  - ML Solutions
- Deep Learning
  - Deep Neural Networks
  - Distributed Deep Learning
  - DL Solutions
- Data Science
- Conclusion

# Three Main Types of Machine Learning




**Courtesy:** https://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/

# Support for Parallel and Distributed Execution

- Scikit-learn:
  - Supports execution via Joblib (https://joblib.readthedocs.io/en/latest/)
  - Joblib supports multi-threaded and multi-process execution (on multiple nodes)
- XGBoost:
  - Multiple ways to run on cluster of nodes:
    - Dask (http://dask.org)
    - Ray (https://ray.io/)
    - AWS YARN
    - Apache Spark (https://spark.apache.org/) using XGBoost4J-Spark
- cuML:
  - Execution is supposed on multiple nodes using Dask (http://dask.org) and NVIDIA's NCCL




# Allreduce Collective Communication Pattern

- Element-wise Sum data from all processes and sends to all processes

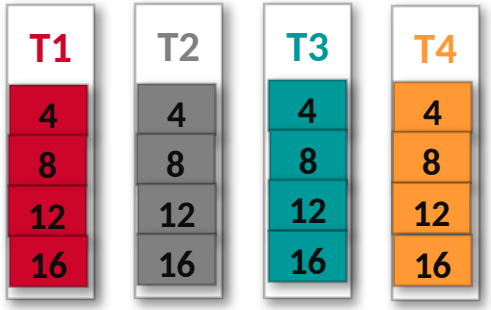
**int MPI_Allreduce (const void *sendbuf, void * recvbuf, int count, MPI_Datatype datatype, MPI_Op operation, MPI_Comm comm)**

| Input-only Parameters | |
|---|---|
| **Parameter** | **Description** |
| sendbuf | Starting address of send buffer |
| recvbuf | Starting address of recv buffer |
| type | Data type of buffer elements |
| count | Number of elements in the buffers |
| operation | Reduction operation to be performed (e.g. sum) |
| comm | Communicator handle |

| Input/Output Parameters | |
|---|---|
| **Parameter** | **Description** |
| recvbuf | Starting address of receive buffer |

**Sendbuf (Before)**


**Recvbuf (After)**



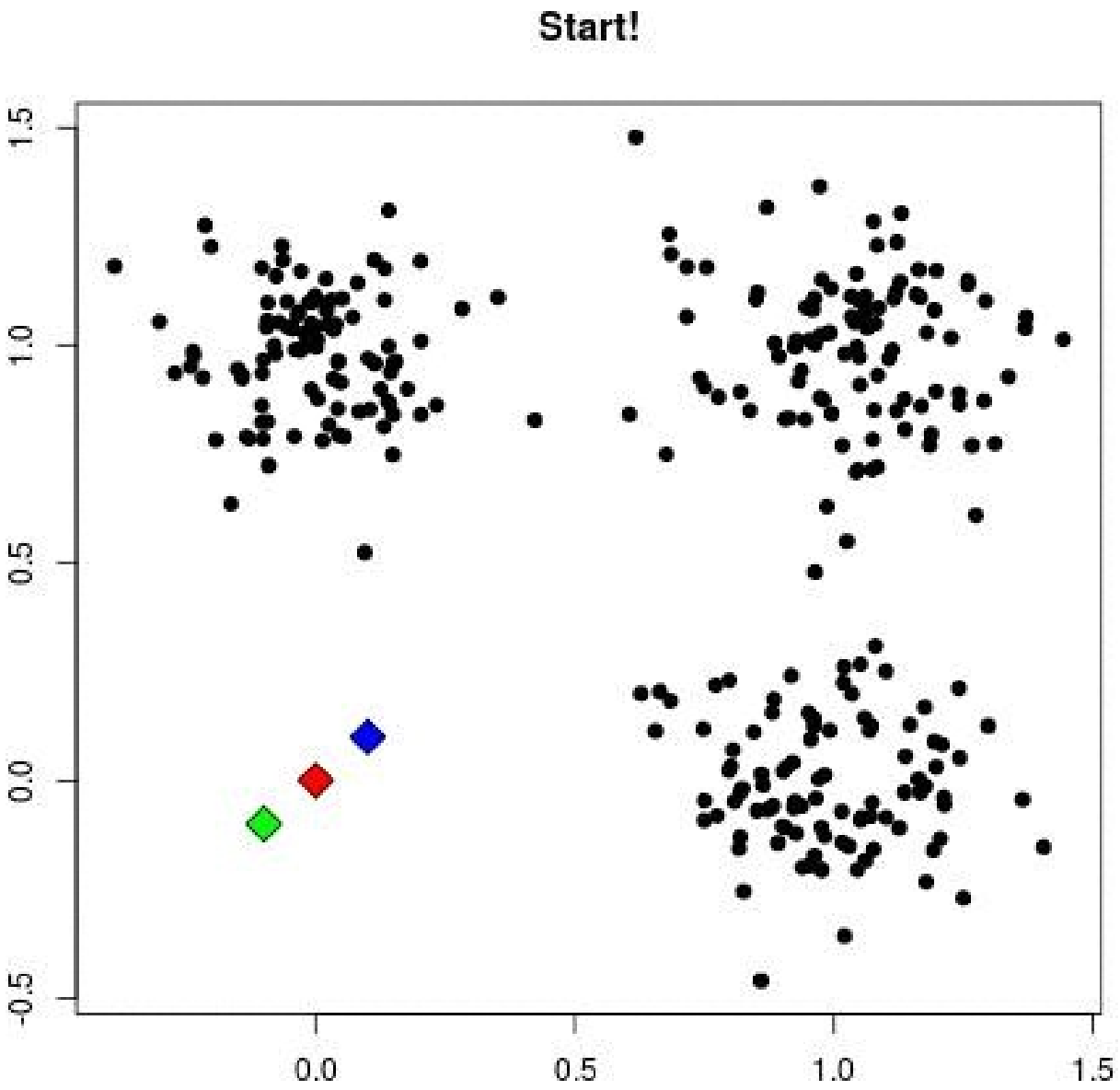
# Overview of the MVAPICH2 Project

- High Performance open-source MPI Library
- Support for multiple interconnects
  - InfiniBand, Omni-Path, Ethernet/iWARP, RDMA over Converged Ethernet (RoCE), AWS EFA, Rockport Networks, and Slingshot10/11, Broadcom
- Support for multiple platforms
  - x86, OpenPOWER, ARM, Xeon-Phi, GPGPUs (NVIDIA and AMD)
- Started in 2001, first open-source version demonstrated at SC '02
- Supports the latest MPI-3.1 standard
- http://mvapich.cse.ohio-state.edu
- Additional optimized versions for different systems/environments:
  - MVAPICH2-X (Advanced MPI + PGAS), since 2011
  - MVAPICH2-GDR with support for NVIDIA (since 2014) and AMD (since 2020) GPUs
  - MVAPICH2-MIC with support for Intel Xeon-Phi, since 2014
  - MVAPICH2-Virt with virtualization support, since 2015
  - MVAPICH2-EA with support for Energy-Awareness, since 2015
  - MVAPICH2-Azure for Azure HPC IB instances, since 2019
  - MVAPICH2-X-AWS for AWS HPC+EFA instances, since 2019
- Tools:
  - OSU MPI Micro-Benchmarks (OMB), since 2003
  - OSU InfiniBand Network Analysis and Monitoring (INAM), since 2015


- **Used by more than 3,275 organizations in 90 countries**
- **More than 1.61 Million downloads from the OSU site directly**
- Empowering many TOP500 clusters (June '22 ranking)
  - 6th, 10,649,600-core (Sunway TaihuLight) at NSC, China
  - 16th, 448, 448 cores (Frontera) at TACC
  - 30th, 288,288 cores (Lassen) at LLNL
  - 42nd, 570,020 cores (Nurion) in South Korea and many more
- Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, OpenHPC, and Spack)
- Partner in the 16th ranked TACC Frontera system
- **Empowering Top500 systems for more than 20 years**

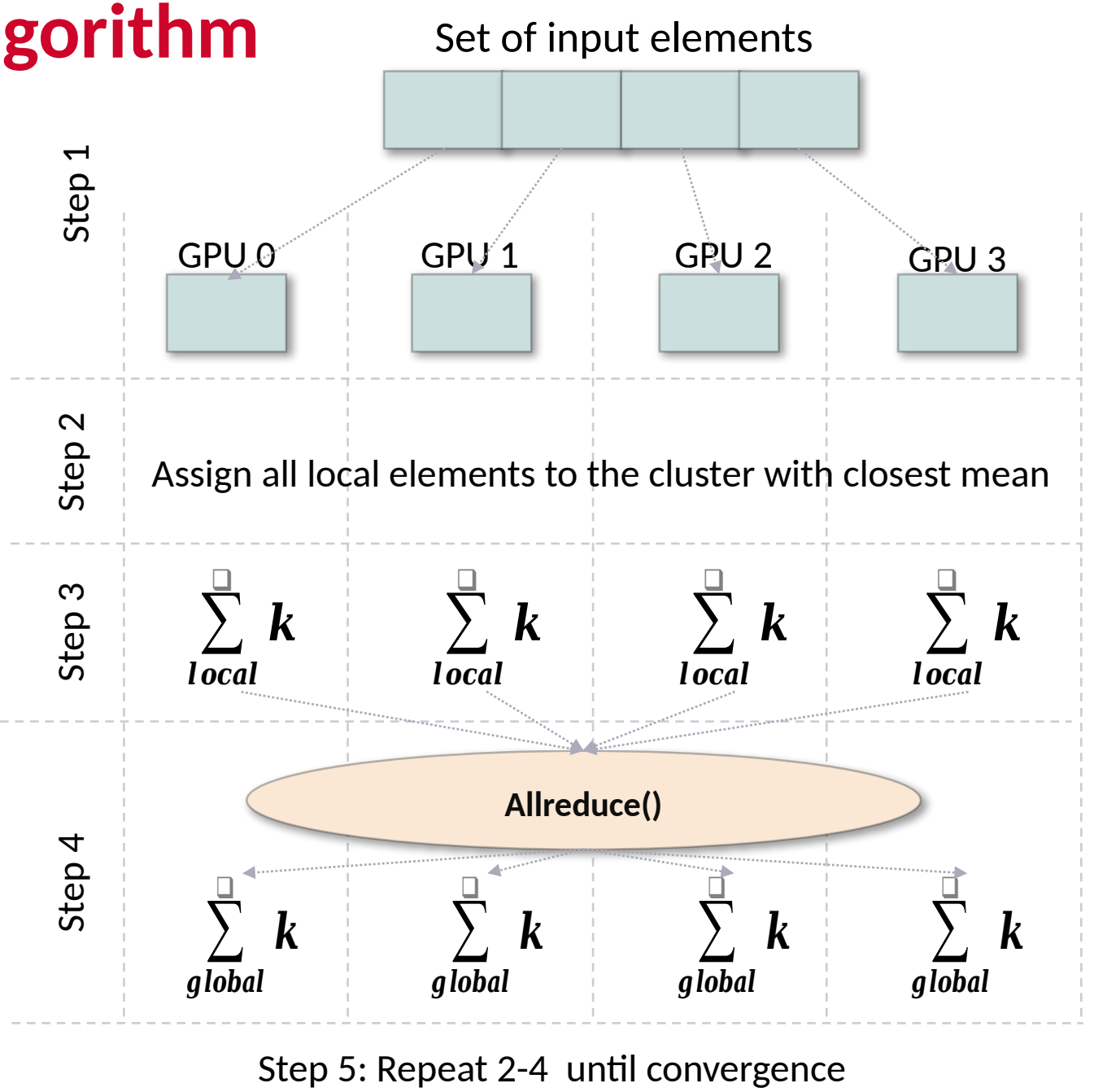
# Parallelizing K-Means

- Step 0: Initialize centroids
  - Assign initial cluster means randomly
- Step 1: Data Division
  - Distribute elements among GPUs
- Step 2: Assign elements (colors)
  - Assign each element to the cluster with the closest mean
- Step 3: Update local cluster mean
  - Calculate local cluster means for all local points
- Step 4: Update global cluster mean*
  - Calculate global cluster means by calling Allreduce()
- Step 5: Repeat steps 2-4 until convergence



# Parallelizing the K-means Algorithm

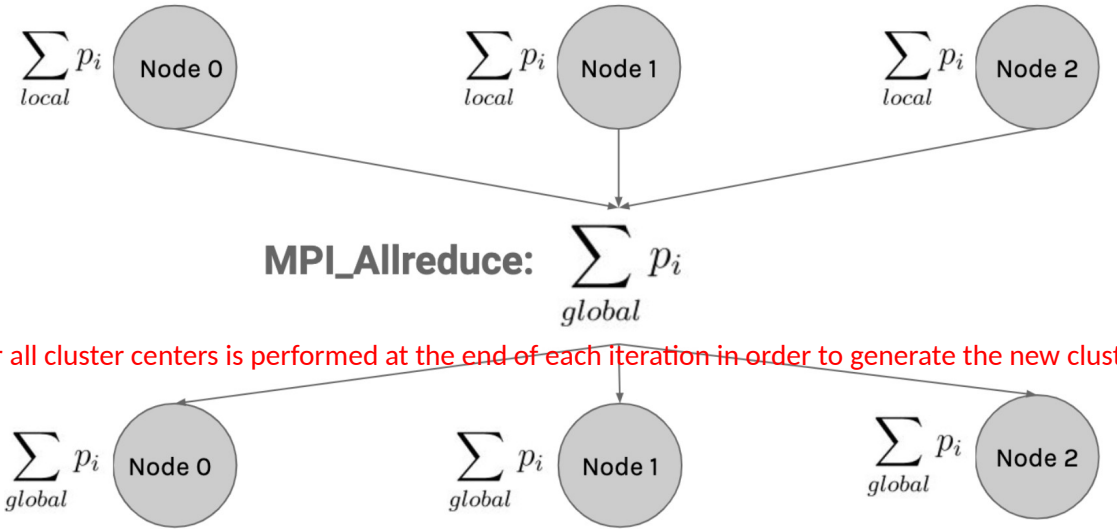
- Step 0: Initialize centroids
  - Assign initial cluster means randomly
- Step 1: Data Division
  - Distribute elements among GPUs
- Step 2: Assign elements (color)
  - Assign each element to the cluster with the closest mean
- Step 3: Update local cluster mean
  - Calculate local cluster means for all local points
- Step 4: Update global cluster mean*
  - Calculate global cluster means by calling Allreduce()
- Step 5: Repeat steps 2-4 until convergence




Step 5: Repeat 2-4 until convergence

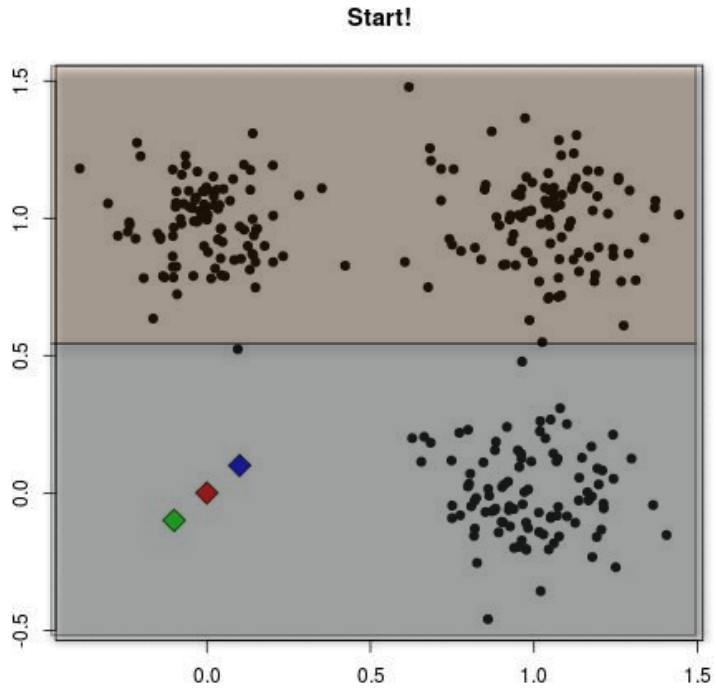
# Parallelizing K-Means Clustering

**Input:**
$D$= {t1, t2, …. Tn } // Set of elements
$K$ // Number of desired clusters
**Output:**
$K$ // Set of clusters
**K-Means algorithm:**
Assign initial values for *m1, m2,…. mk*
**repeat**
assign each item ti to the clusters which has the closest mean;
calculate new mean for each cluster;
**until** convergence criteria is met;




Global reduction for all cluster centers is performed at the end of each iteration in order to generate the new cluster centers.

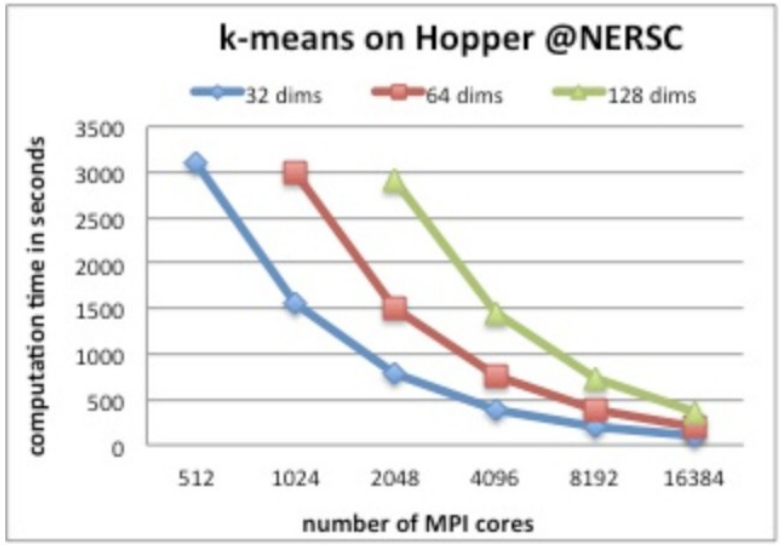
Recompute centroids after MPI_Allreduce




Node 0

Node 1

Domain Decomposition




**Courtesy:** https://github.com/tmscarla/k-means-parallel http://users.eecs.northwestern.edu/~wkliao/Kmeans/

# Outline

- Introduction
- **Machine Learning**
  - Distributed K-Means
  - **ML Solutions**
- Deep Learning
  - Deep Neural Networks
  - Distributed Deep Learning
  - DL Solutions
- Data Science
- Conclusion

# The cuML Library: Accelerating ML on GPUs

- The NVIDIA RAPIDS project aims to build end-to-end data science analytic pipelines on GPUs
- An important component is the cuML library:
  - GPU-accelerated ML library
  - GPU-counterpart of Scikit-learn
  - Supports the execution of ML workloads on Multi-Node Multi-GPUs (MNMG) systems
- Most existing ML libraries, including Scikit-learn and Apache Spark's MLlib, only support CPU execution of ML algorithms
  - Conventional wisdom has been that only DNNs are a good match for GPUs because of high computational requirements

# Main components of the cuML library

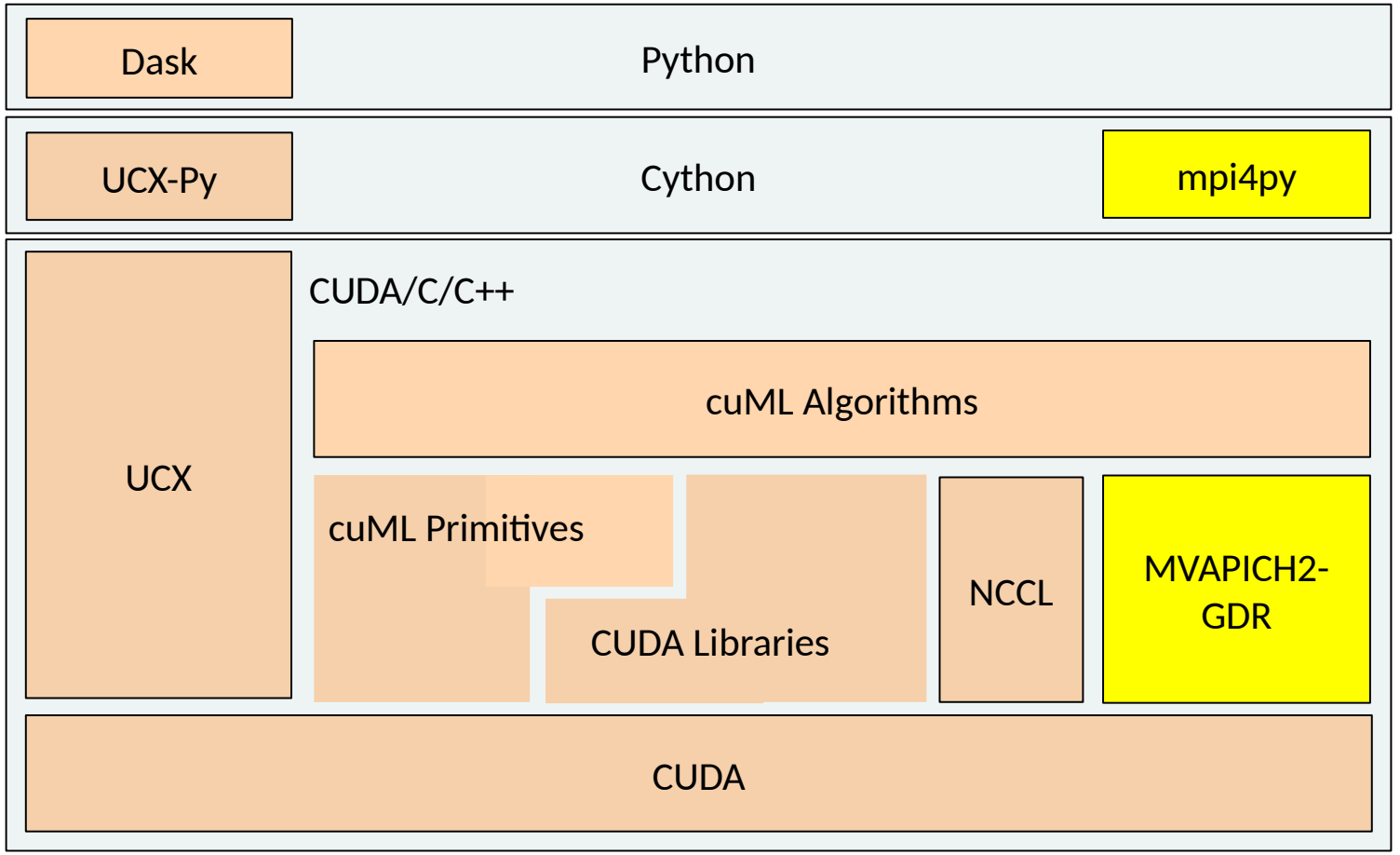
- Main components
  - Python layer
    - Provides a Scikit-learn like interface
    - Hides the complexities of the CUDA/C/C++ layer
  - Primitives and cuML algorithms built on top of CUDA
    - ML Algorithms
    - Primitives
      - Reusable building blocks for building machine learning algorithms
      - Common for different machine learning algorithms
      - Used to build different machine learning algorithms
  - Communication Support in cuML:
    - Point-to-point communication: Dask
    - Collective communication: NVIDIA Collective Communications Library (NCCL)

# Accelerating cuML with MVAPICH2-GDR

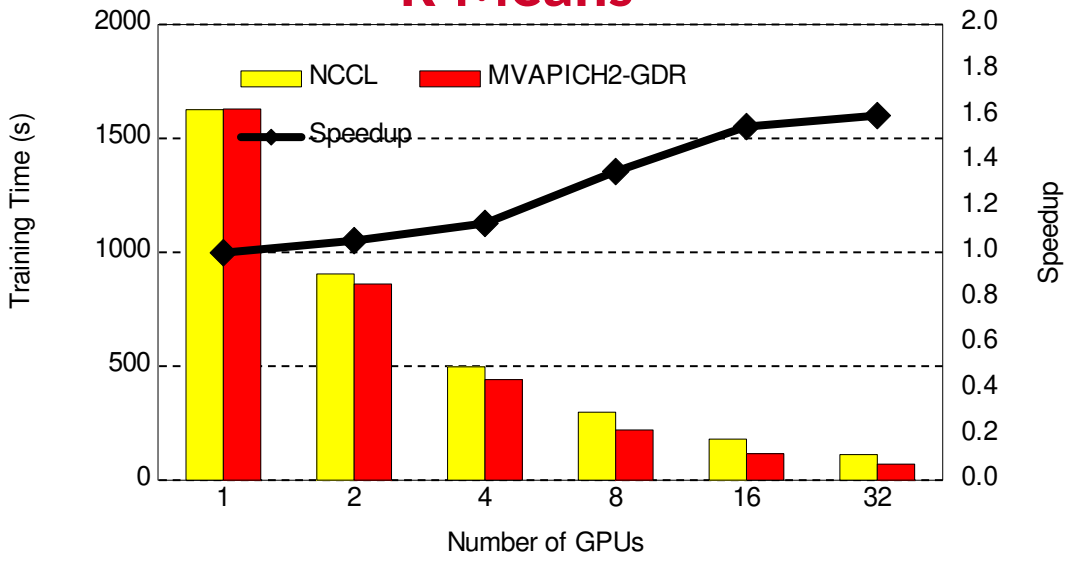
- Utilize MVAPICH2-GDR (with mpi4py) as communication backend during the training phase (the fit() function) in the Multi-node Multi-GPU (MNMG) setting over cluster of GPUs
- Communication primitives:
  - Allreduce
  - Reduce
  - Broadcast
- Exploit optimized collectives

**MPI4cuML 0.1 release**

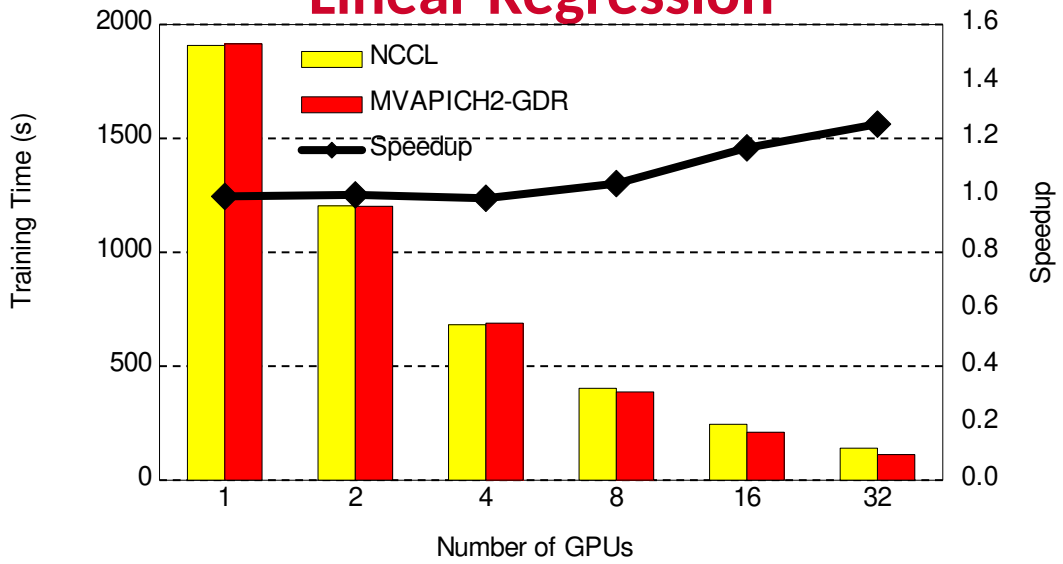
**(http://hidl.cse.ohio-state.edu)**



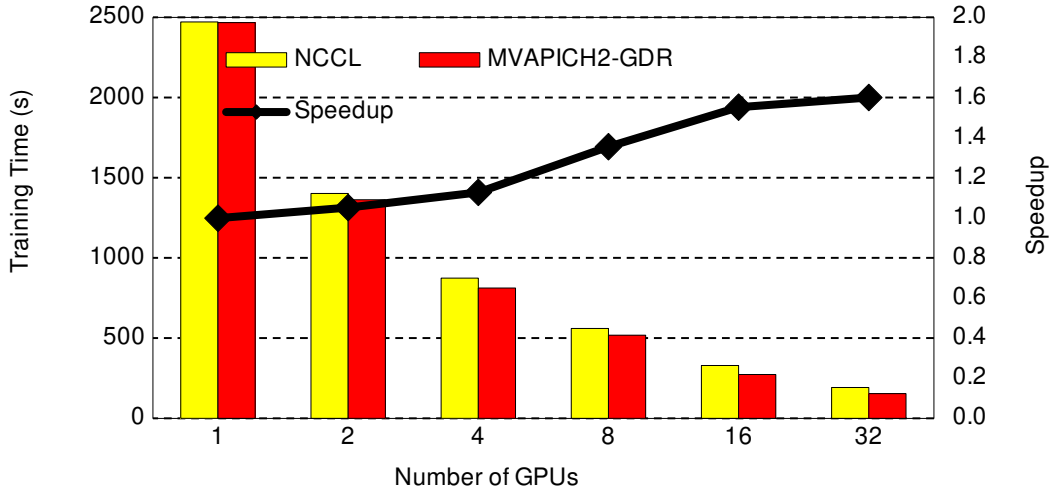
## K-Means



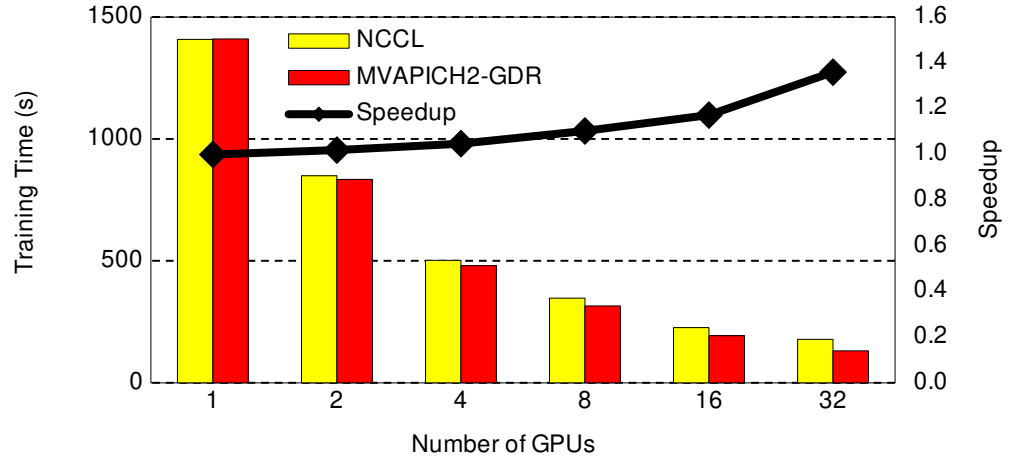

## Linear Regression




## Nearest Neighbors




## Truncated SVD



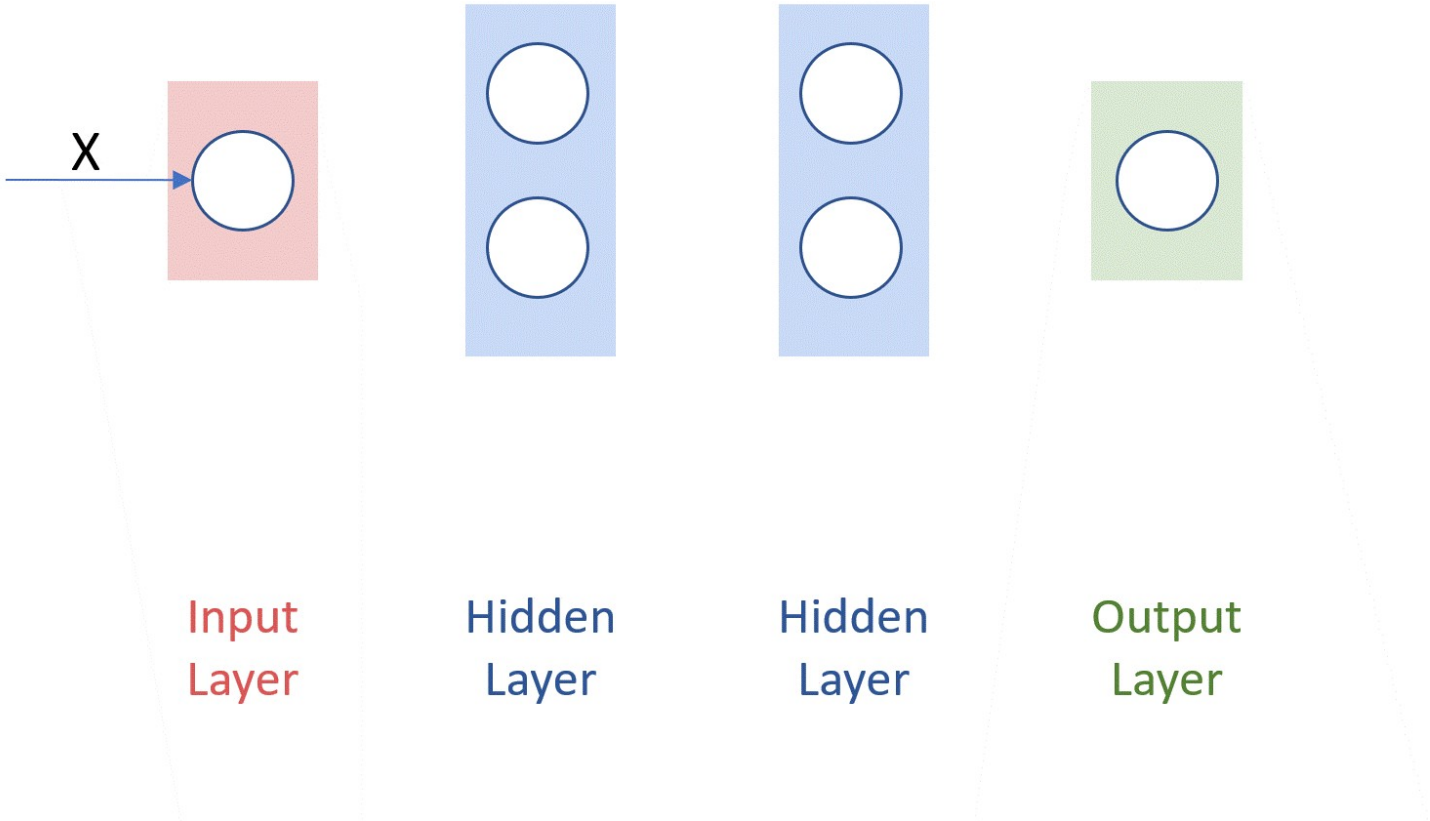

M. Ghazimirsaeed , Q. Anthony , A. Shafi , H. Subramoni , and D. K. Panda, Accelerating GPU-based Machine Learning in Python using MPI Library: A Case Study with MVAPICH2-GDR, MLHPC Workshop, Nov 2020

# Outline

- Introduction
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- **Deep Learning**
  - **Deep Neural Networks**
  - Distributed Deep Learning
  - DL Solutions
- Data Science
- Conclusion

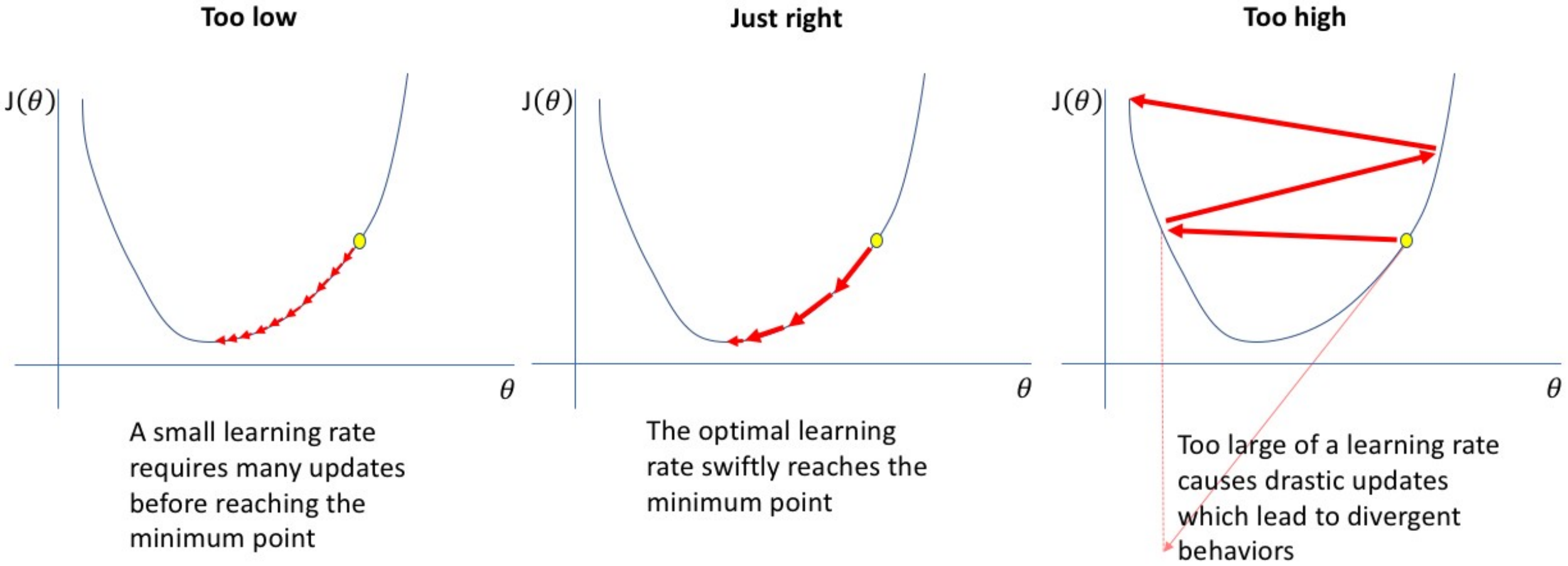
# Understanding the Deep Neural Network Concepts

- Example of a 3-layer Deep Neural Network (DNN) – (input layer is not counted)




**Courtesy:** http://cs231n.github.io/neural-networks-1/

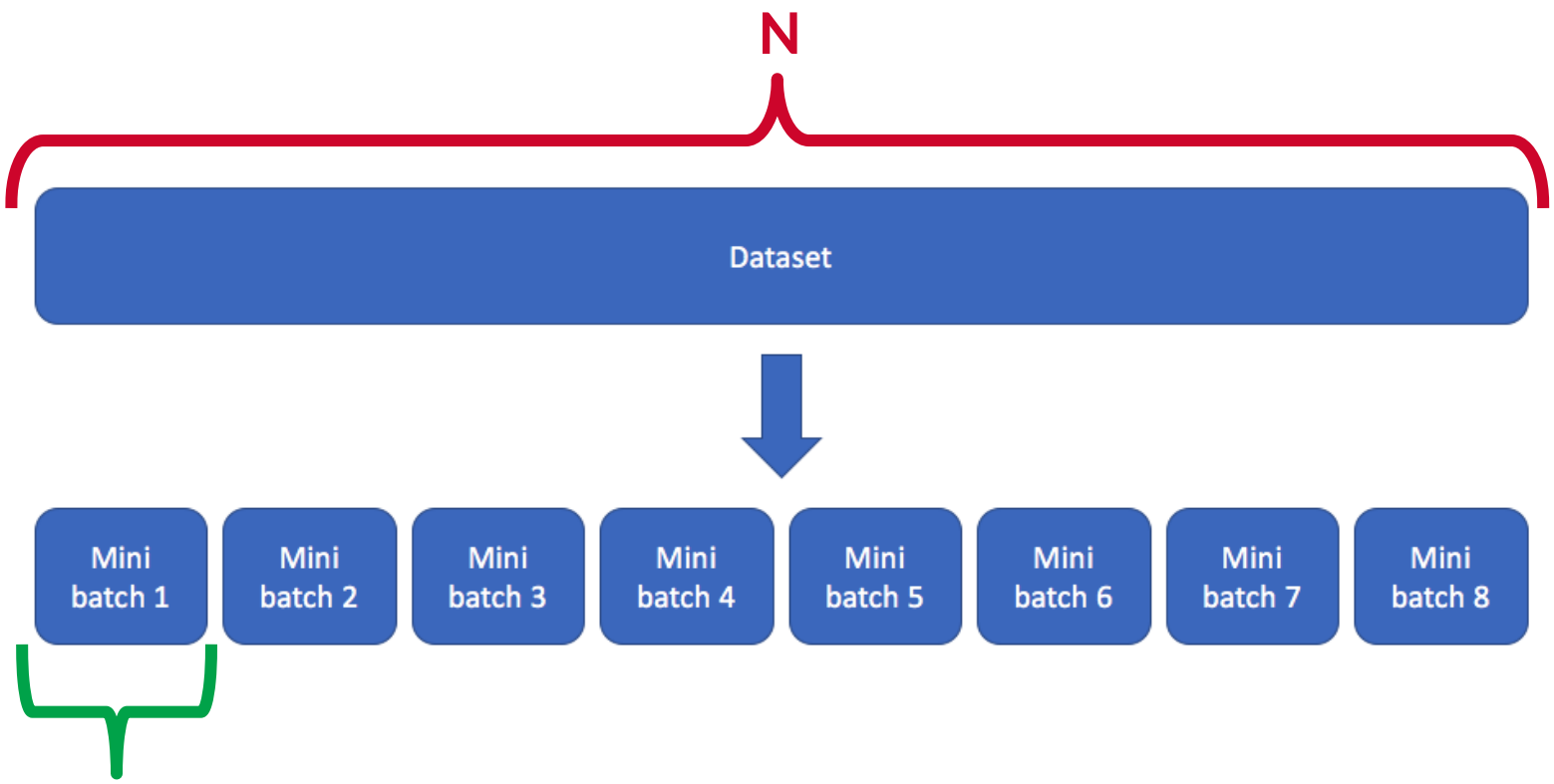
# Essential Concepts: Learning Rate (α)




**Courtesy:** https://www.jeremyjordan.me/nn-learning-rate/

# Essential Concepts: Batch Size

- Batched Gradient Descent
  - Batch Size = **N**
- Stochastic Gradient Descent
  - Batch Size = **1**
- Mini-batch Gradient Descent
  - Somewhere in the middle
  - Common:
    - Batch Size = 64, 128, 256, etc.
- Finding the optimal batch size will yield the fastest learning.




*One full pass over **N** is called an **epoch** of training*

**Courtesy:** https://www.jeremyjordan.me/gradient-descent/

# Outline

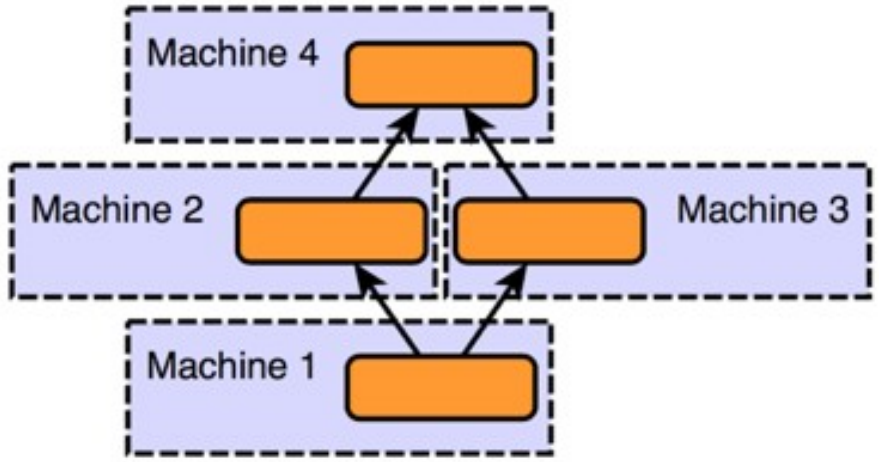
- Introduction
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- **Deep Learning**
  - Deep Neural Networks
  - **Distributed Deep Learning**
  - DL Solutions
- Data Science
- Conclusion

# The Need for Parallel and Distributed Training

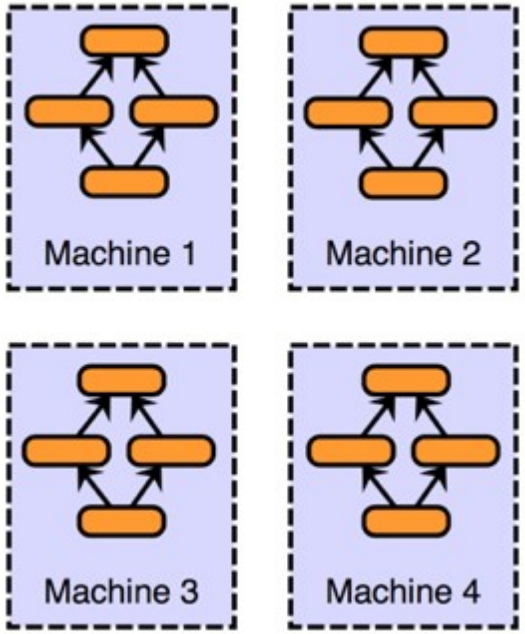
- Why do we need Parallel Training?
- Larger and Deeper models are being proposed
  - **AlexNet -> ResNet -> NASNet – AmoebaNet**
  - DNNs require a lot of memory and a lot of computation
  - Larger models cannot fit a GPU's memory
- Single GPU training cannot keep up with ever-larger models
- Community has moved to multi-GPU training
- Multi-GPU in one node is good but there is a limit to Scale-up (8-16 GPUs)
- **Multi-node (Distributed or Parallel) Training is necessary!!**

# Parallelization Strategies

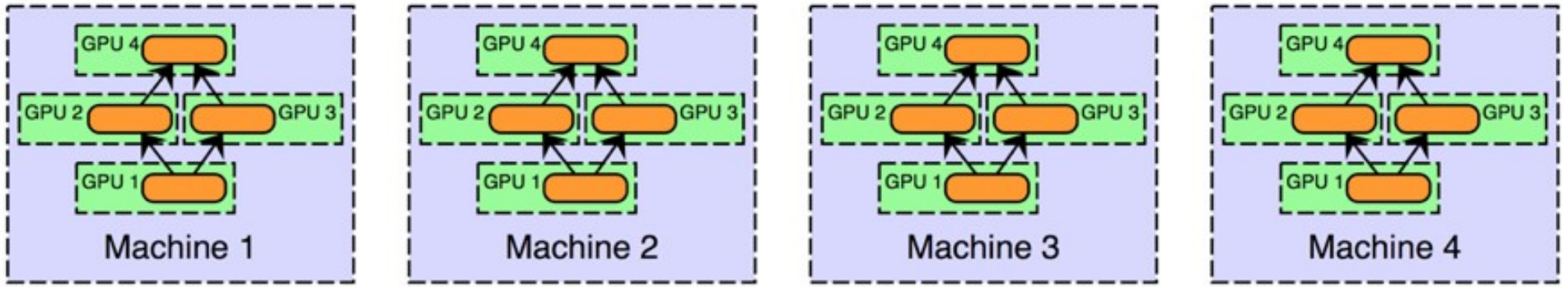
- Some parallelization strategies..
  - Data Parallelism or Model Parallelism
  - Hybrid Parallelism




**Model Parallelism**




**Data Parallelism**

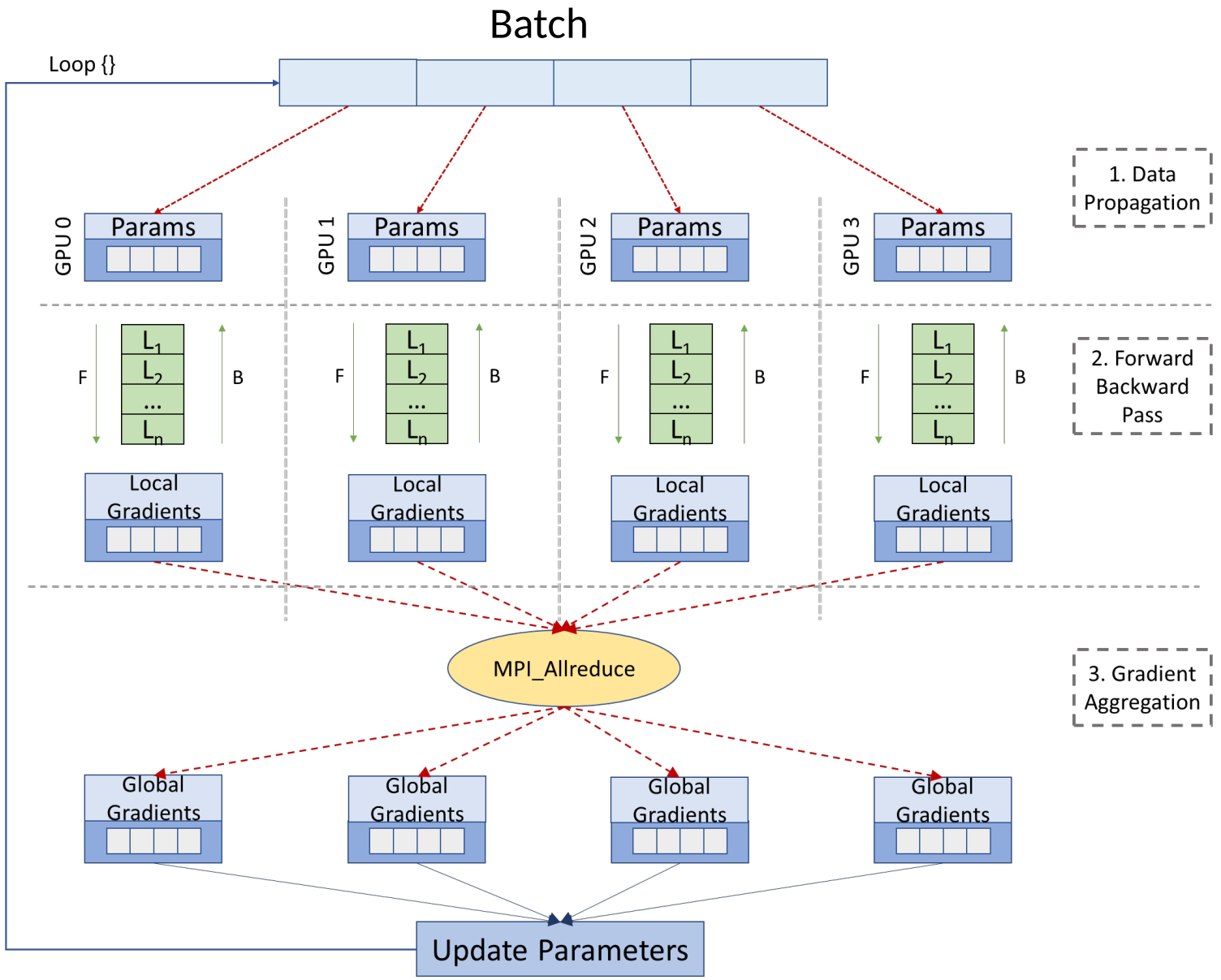



**Hybrid (Model and Data) Parallelism**

**Courtesy:** http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks

# Data Parallelism and MPI Collectives

- Step1: Data Propagation
  - Distribute the Data among GPUs
- Step2: Forward Backward Pass
  - Perform forward pass and calculate the prediction
  - Calculate Error by comparing prediction with actual output
  - Perform backward pass and calculate gradients
- Step3: Gradient Aggregation
  - Call MPI_Allreduce to reduce the local gradients
  - Update parameters locally using global gradients

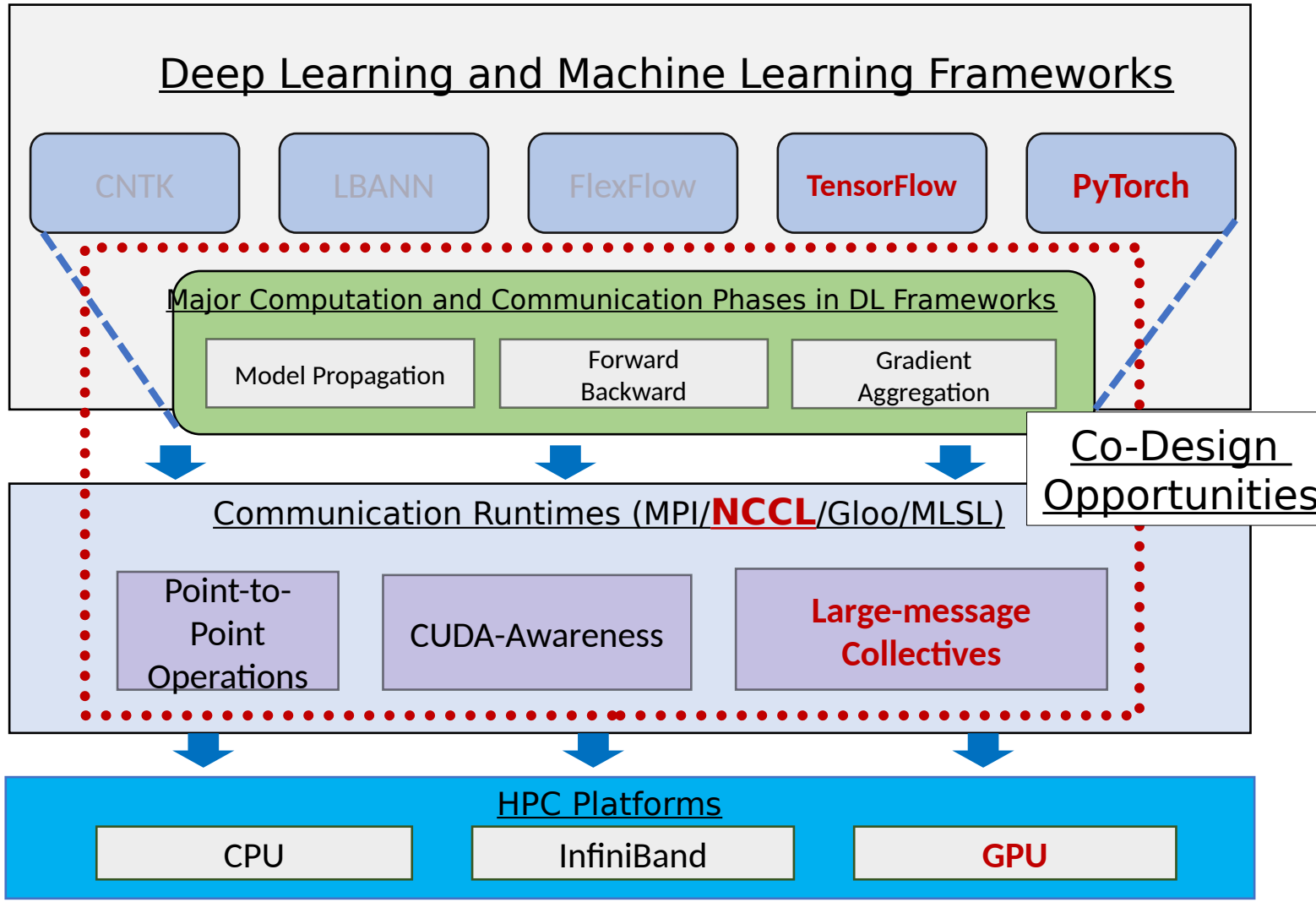


# Outline

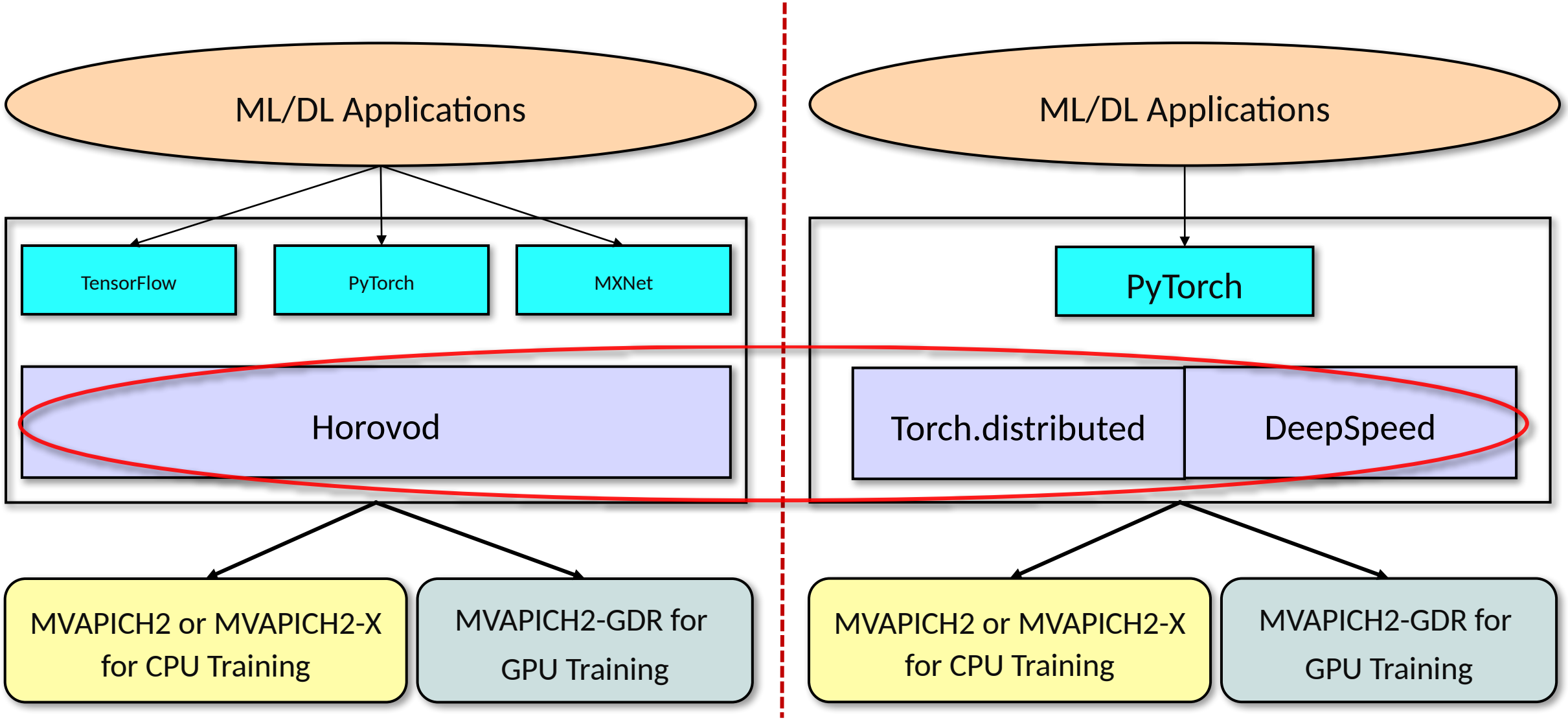
- Introduction
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- **Deep Learning**
  - Deep Neural Networks
  - Distributed Deep Learning
  - **DL Solutions**
- Data Science
- Conclusion

# Solutions and Case Studies: Exploiting HPC for DL

- **Data Parallelism**
- Model-Parallelism



# MVAPICH2 (MPI)-driven Infrastructure for ML/DL Training




**More details available from: http://hidl.cse.ohio-state.edu**

# Install Horovod with MVAPICH2-X and MVAPICH2-GDR

## Command to install Horovod with MVAPICH2-X

```
$ HOROVOD_WITH_MPI=1 pip install --no-cache-dir horovod
```

## Command to install Horovod with MVAPICH2-GDR

```
$ HOROVOD_GPU_ALLREDUCE=MPI HOROVOD_CUDA_HOME=/opt/cuda/11.3 HOROVOD_WITH_MPI=1 pip
install --no-cache-dir horovod
```

# Run PyTorch on a single GPU

+ python pytorch_synthetic_benchmark.py --batch-size 64 --num-iters=5

.

.

V100

```
Model: resnet50
Batch size: 64
Number of GPUs: 1
Running warmup...
Running benchmark...
Iter #0: 333.9 img/sec per GPU
Iter #1: 334.2 img/sec per GPU
Iter #2: 333.9 img/sec per GPU
Iter #3: 333.8 img/sec per GPU
Iter #4: 333.9 img/sec per GPU
Img/sec per GPU: 334.0 +-0.2
----------------------------------------
Total img/sec on 1 GPU(s): 334.0 +-0.2
----------------------------------------
```

# Run PyTorch on two nodes with 1 GPU/node (using MVAPICH2-GDR)

+ mpirun_rsh -np 2 gpu11 gpu12 MV2_USE_CUDA=1 MV2_CPU_BINDING_POLICY=hybrid MV2_HYBRID_BINDING_POLICY=spread MV2_USE_RDMA_CM=0 MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so LD_PRELOAD=mv2-gdr/lib/libmpi.so python pytorch_synthetic_benchmark.py --batch-size 64 --num-iters=5

V100

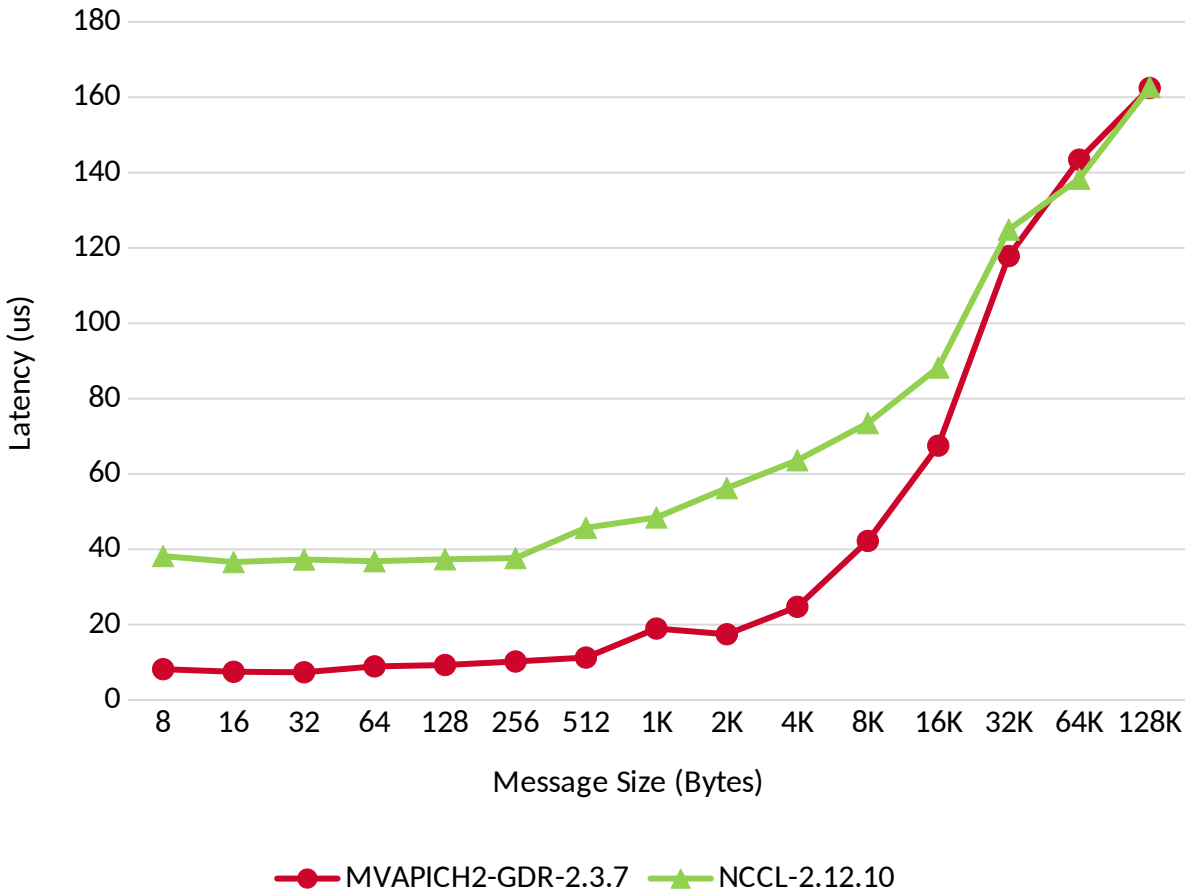
```
.
.
Model: resnet50
Batch size: 64
Number of GPUs: 2
Running warmup...
Running benchmark...
Iter #0: 317.0 img/sec per GPU
Iter #1: 314.9 img/sec per GPU
Iter #2: 315.4 img/sec per GPU
Iter #3: 318.0 img/sec per GPU
Iter #4: 316.7 img/sec per GPU
Img/sec per GPU: 316.4 +-2.2
-----------------------------------
Total img/sec on 2 GPU(s): 632.8 +-4.3
-----------------------------------
```

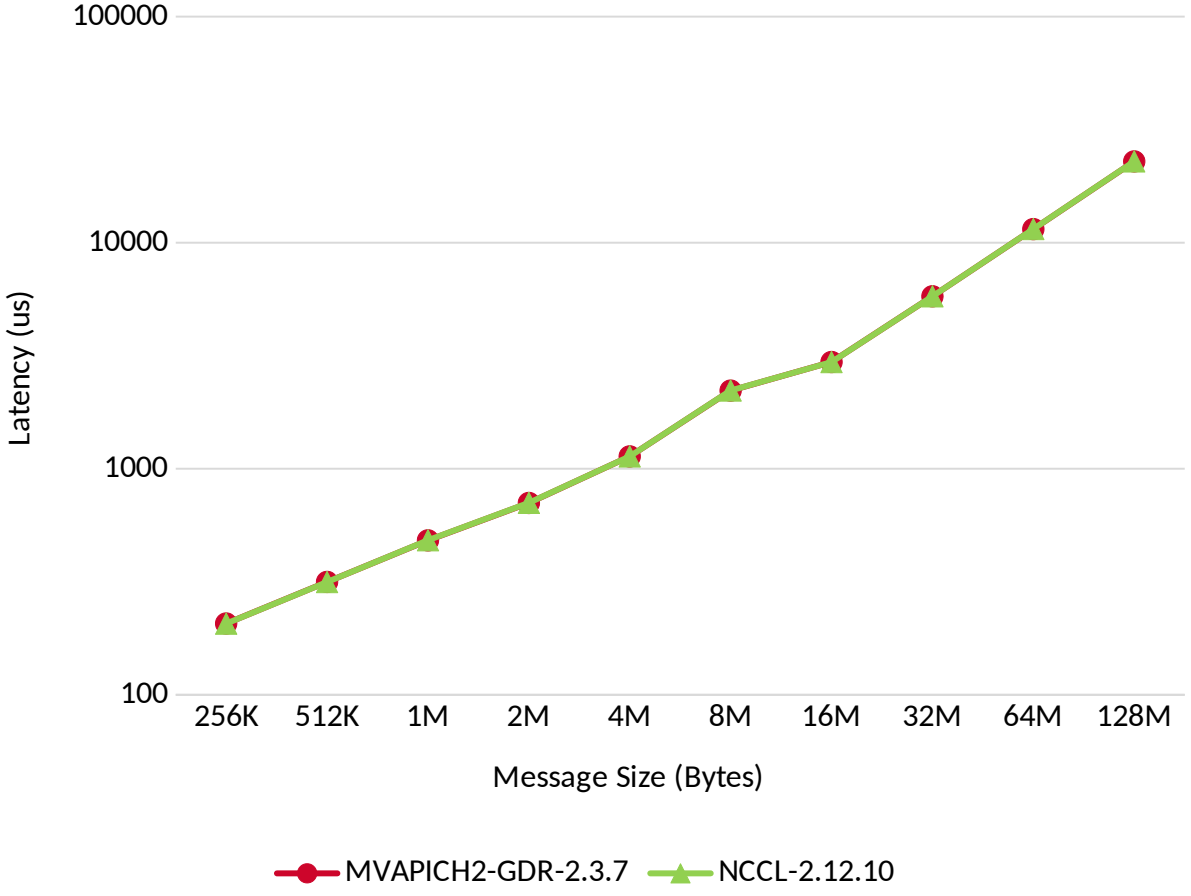
~1.89X on 2 GPUs

# MVAPICH2-GDR vs. NCCL2 – Allreduce Operation (OSC Pitzer)

- Optimized designs in MVAPICH2-GDR offer better/comparable performance for most cases
- MPI_Allreduce (MVAPICH2-GDR) vs. ncclAllreduce (NCCL2) on OSC Pitzer system

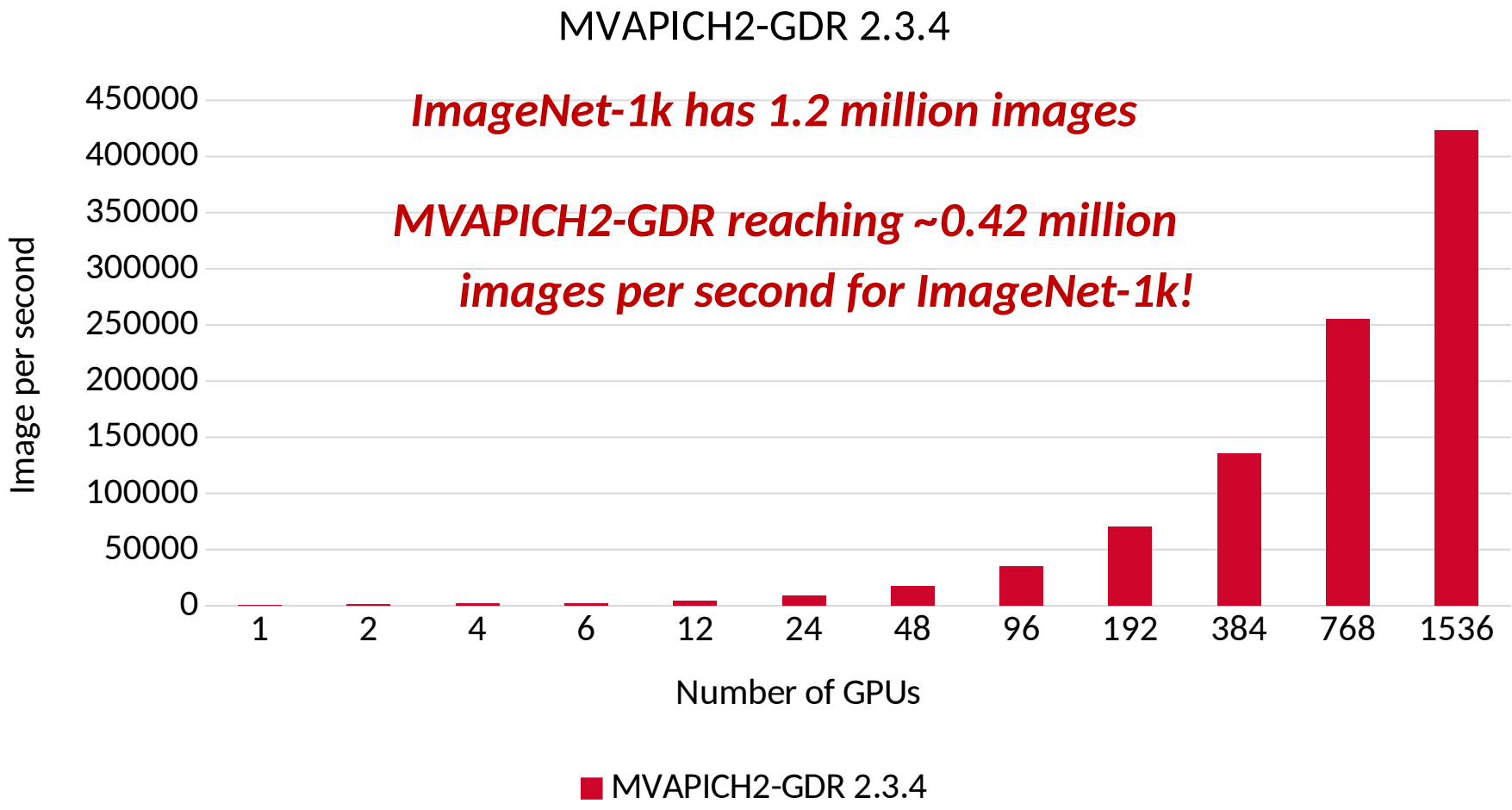
*Platform: OSC Pitzer system (8 nodes with 2 Nvidia Volta GPUs), CUDA 11.6*



# Distributed TensorFlow on ORNL Summit (1,536 GPUs)

- ResNet-50 Training using TensorFlow benchmark on SUMMIT -- 1536 Volta GPUs!
- 1,281,167 (1.2 mil.) images
- Time/epoch = 3 seconds
- Total Time (90 epochs) = 3 x 90 = 270 seconds = **4.5 minutes!**

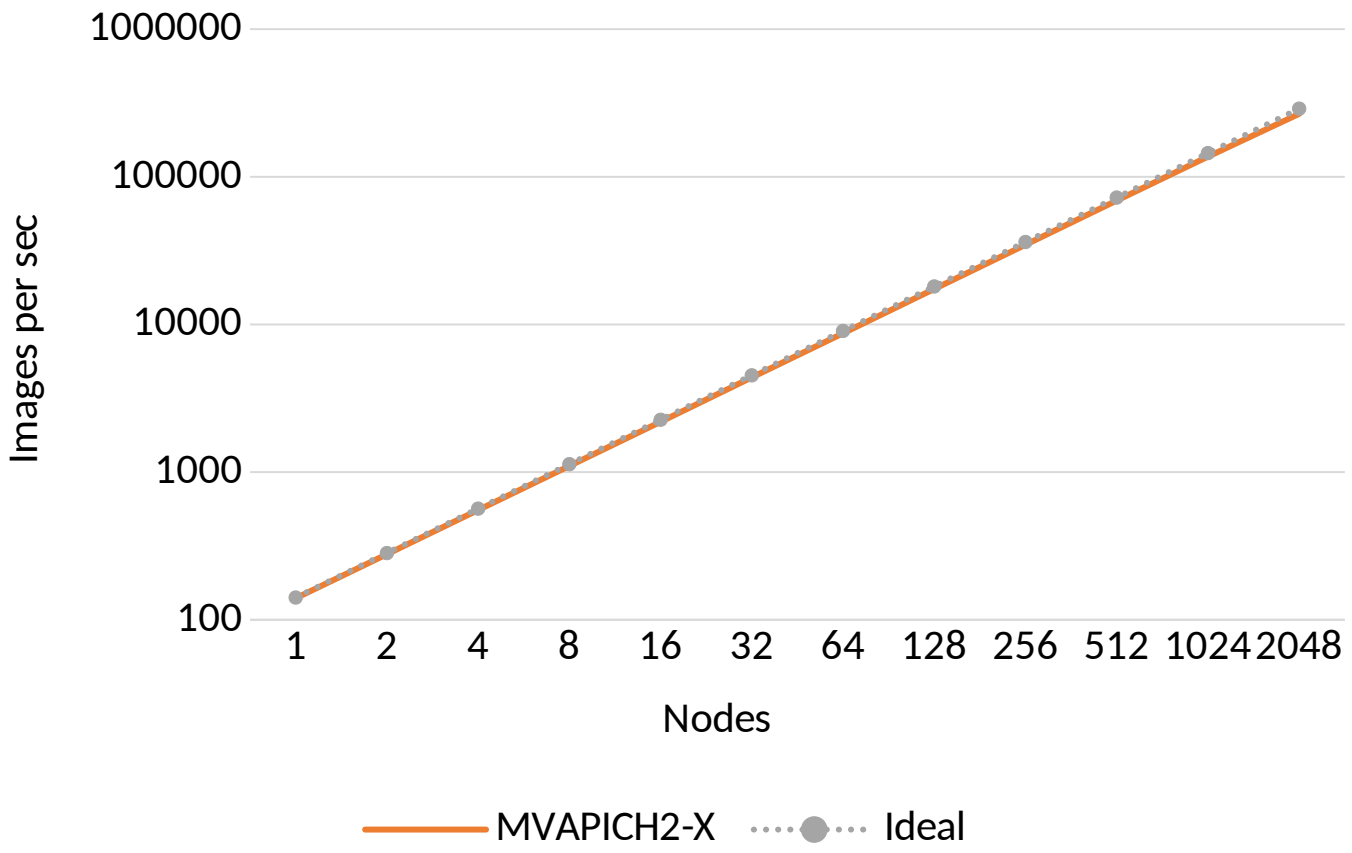



*We observed issues for NCCL2 beyond 384 GPUs

*Platform: The Summit Supercomputer (#2 on Top500.org) – 6 NVIDIA Volta GPUs per node connected with NVLink, CUDA 10.1*

# Distributed TensorFlow on TACC Frontera (2048 CPU nodes)

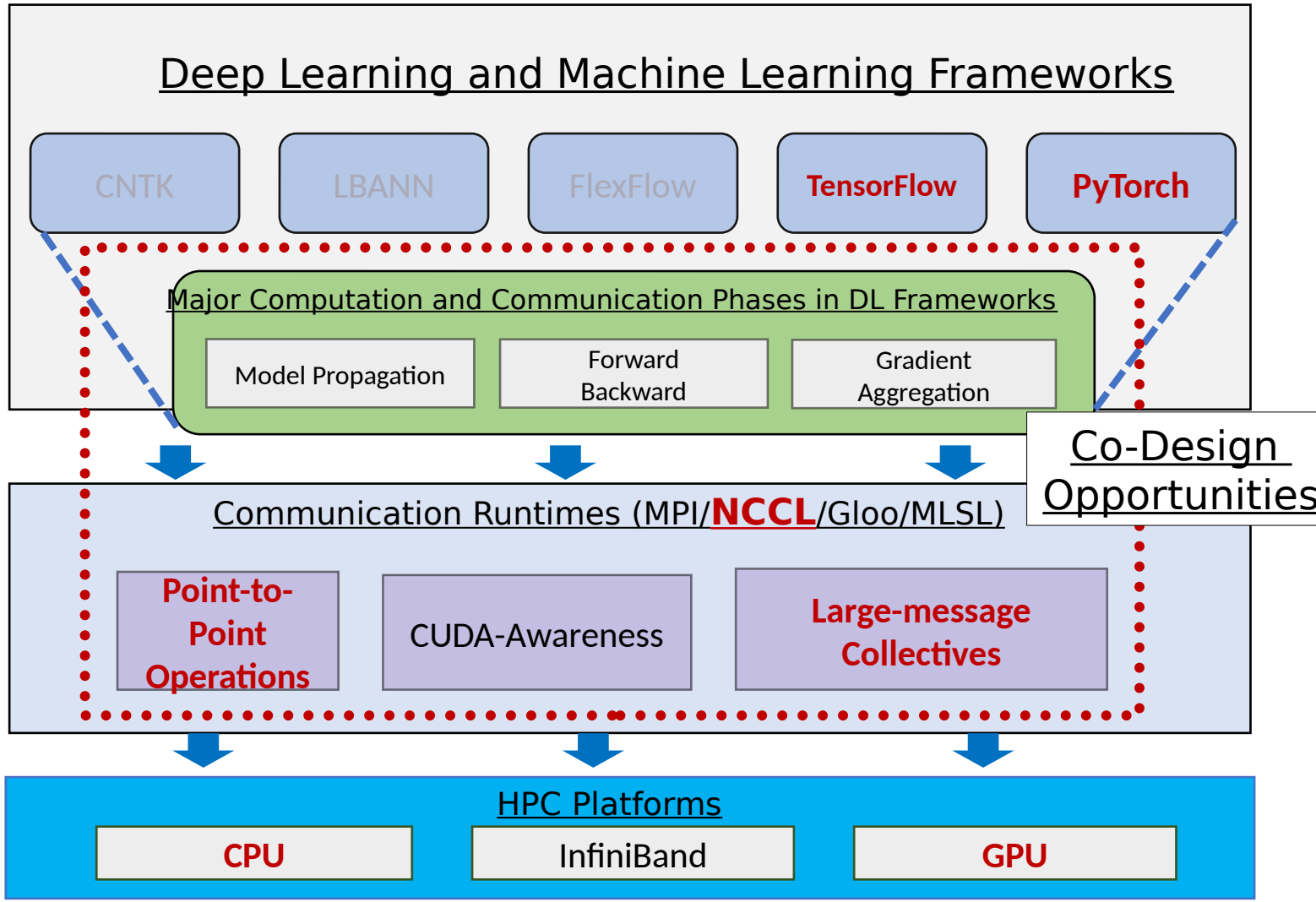
- Scaled TensorFlow to 2048 nodes on Frontera using MVAPICH2 and IntelMPI
- MVAPICH2 delivers close to the ideal performance for DNN training
- Report a peak of 260,000 images/sec on 2048 nodes
- On 2048 nodes, ResNet-50 can be trained in 7 minutes!




A. Jain, A. A. Awan, H. Subramoni, DK Panda, "Scaling TensorFlow, PyTorch, and MXNet using MVAPICH2 for High-Performance Deep Learning on Frontera", DLS '19 (SC '19 Workshop).

# Solutions and Case Studies: Exploiting HPC for DL

- Data Parallelism
- **Model-Parallelism**



# HyPar-Flow: Hybrid Parallelism for TensorFlow

- Why Hybrid parallelism?
  - Data Parallel training has limits! 
- We propose HyPar-Flow
  - An easy to use Hybrid parallel training framework
    - Hybrid = Data + Model
  - Supports Keras models and exploits TF 2.0 Eager Execution
  - Exploits MPI for Point-to-point and Collectives

***Benchmarking large-models lead to better insights and ability to develop new approaches!***

**A. A. Awan, A. Jain, Q. Anthony, H. Subramoni, and DK Panda, "HyPar-Flow: Exploiting MPI and Keras for Hybrid Parallel Training of TensorFlow models", ISC '20,** https://arxiv.org/pdf/1911.05146.pdf

# Model/Hybrid Parallelism and MPI Collectives

- HyPar-Flow is practical (easy-to-use) and high-performance (uses MPI)
  - Based on Keras models and exploits TF 2.0 Eager Execution
  - Leverages MPI Pt-to-pt. and Collectives for communication

**A. A. Awan, A. Jain, Q. Anthony, H. Subramoni, and DK Panda, "HyPar-Flow: Exploiting MPI and Keras for Hybrid Parallel Training of TensorFlow models", ISC '20, https://arxiv.org/pdf/1911.05146.pdf**

# HyPar-Flow at Scale (512 nodes on TACC Frontera)

- ResNet-1001 with variable batch size
- Approach:
  - 48 model-partitions for 56 cores
  - 512 model-replicas for 512 nodes
  - Total cores: 48 x 512 = 24,576
- Speedup
  - **253X** on 256 nodes
  - **481X** on 512 nodes
- Scaling Efficiency
  - **98%** up to 256 nodes
  - **93.9%** for 512 nodes

**481x speedup on 512 Intel Xeon Cascade Lake nodes (TACC Frontera)**

**A. A. Awan, A. Jain, Q. Anthony, H. Subramoni, and DK Panda, "HyPar-Flow: Exploiting MPI and Keras for Hybrid Parallel Training of TensorFlow models", ISC '20**, https://arxiv.org/pdf/1911.05146.pdf

# GEMS: GPU Enabled Memory Aware Model Parallelism Systems

Why do we need Memory aware designs?

- Data and Model Parallel training has limitation!

- Maximum Batch Size depends on the memory.

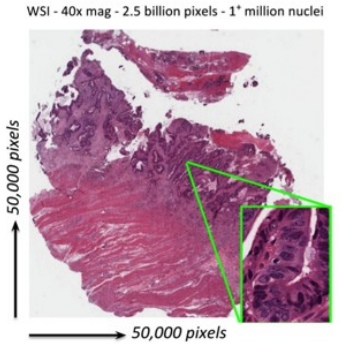
- Basic Model Parallelism suffers from underutilization of memory and compute 

***Memory requirement increases with the increase in image size!***

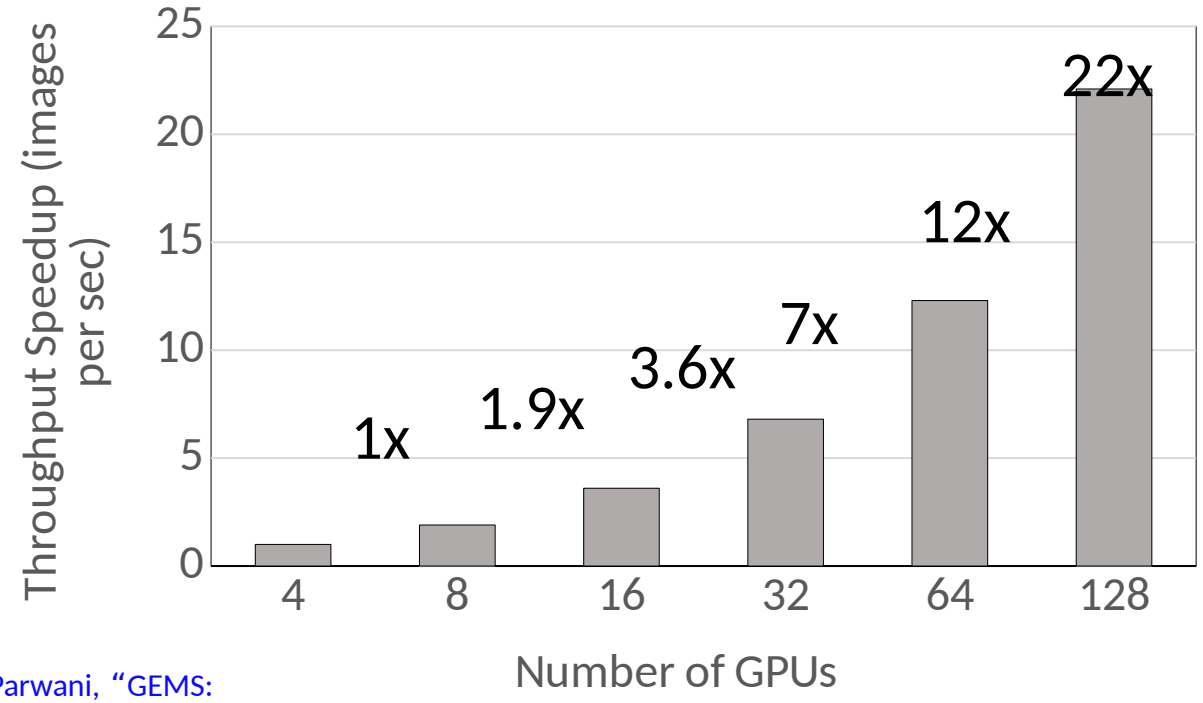
A. Jain, A. Awan, A. Aljuhani, J. Hashmi, Q. Anthony, H. Subramoni, D. Panda, R. Machiraju, A. Parwani, "GEMS: GPU Enabled Memory Aware Model Parallelism System for Distributed DNN", SC '20.

# Exploiting Model Parallelism in AI-Driven Digital Pathology

- Pathology whole slide image (WSI)
  - Each WSI = 100,000 x 100,000 pixels
  - Can not fit in a single GPU memory
  - Tiles are extracted to make training possible
- Two main problems with tiles
  - Restricted tile size because of GPU memory limitation
  - Smaller tiles loose structural information
- Reduced training time significantly
  - **GEMS-Basic: 7.25 hours (1 node, 4 GPUs)**
  - **GEMS-MAST: 6.28 hours (1 node, 4 GPUs)**
  - **GEMS-MASTER: 4.21 hours (1 node, 4 GPUs)**
  - **GEMS-Hybrid: 27 mins (32 nodes, 128 GPUs)**
  - **Overall 15x reduction in training time!!!!**




**Courtesy:**
https://blog.kitware.com/digital-slide-archive-large-image-and-histomicstk-open-source-informatics-tools-for-management-visualization-and-analysis-of-digital-histopathology-data/




**Scaling ResNet110 v2 on 1024×1024 image tiles using histopathology data**

A. Jain, A. Awan, A. Aljuhani, J. Hashmi, Q. Anthony, H. Subramoni, D. K. Panda, R. Machiraju, and A. Parwani, "GEMS: GPU Enabled Memory Aware Model Parallelism System for Distributed DNN Training", Supercomputing (SC '20).

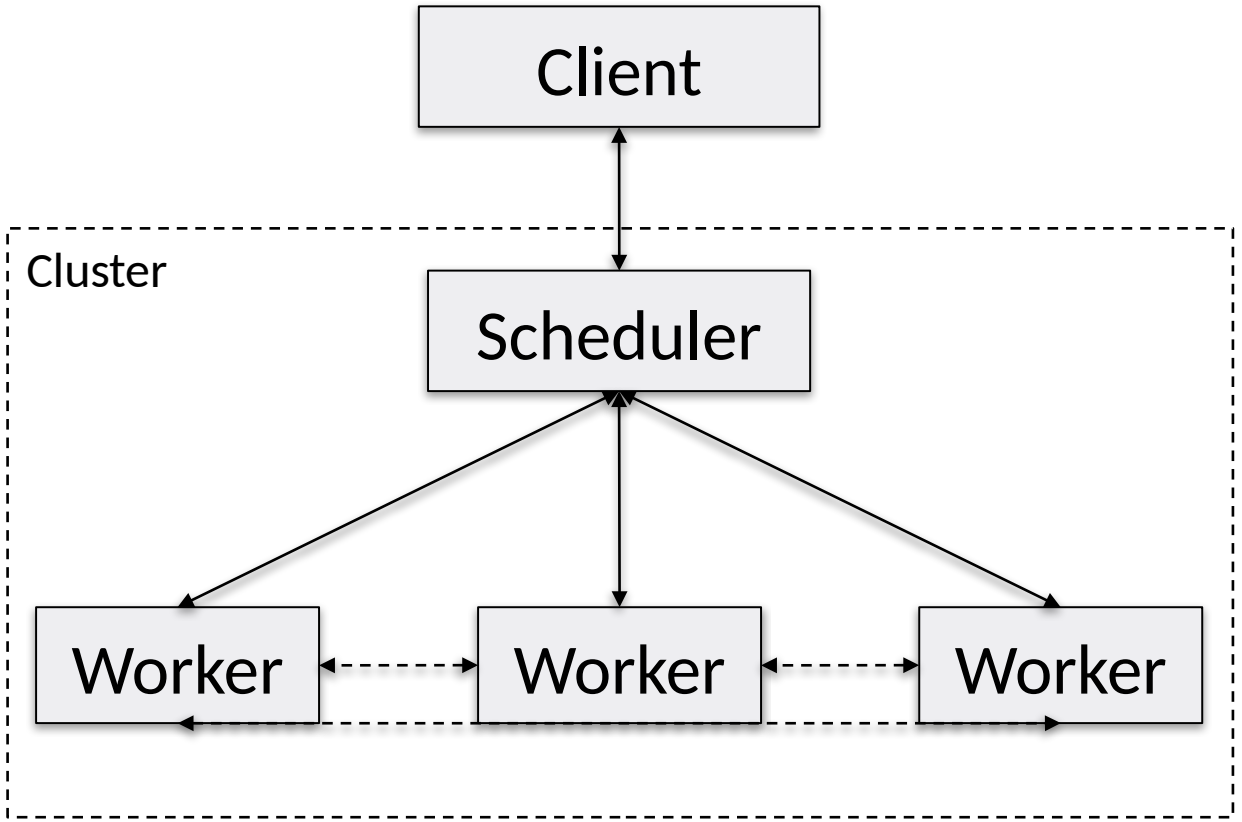
# Outline

- Introduction
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- Deep Learning
  - Deep Neural Networks
  - Distributed Deep Learning
  - DL Solutions
- **Data Science**
- Conclusion

# Introduction to Dask

- Dask is a popular task-based distributed computing framework:
  - Scales Python applications from laptops to high-end systems
  - Builds a task-graph that is executed lazily on parallel hardware
  - Natively extends popular data processing libraries like numPy, Pandas
- Dask Distributed library supports parallel and distributed execution:
  - Built using the asyncio package that allows execution of asynchronous/non-blocking/concurrent operations called *coroutines:*
    - These are defined using async and invoked using await
  - Dask Distributed library provide support for communication:
    - TCP: Tornado-based
    - UCX: Built using a Cython wrapper called UCX-Py
- Other Data Science frameworks include Apache Spark and Ray

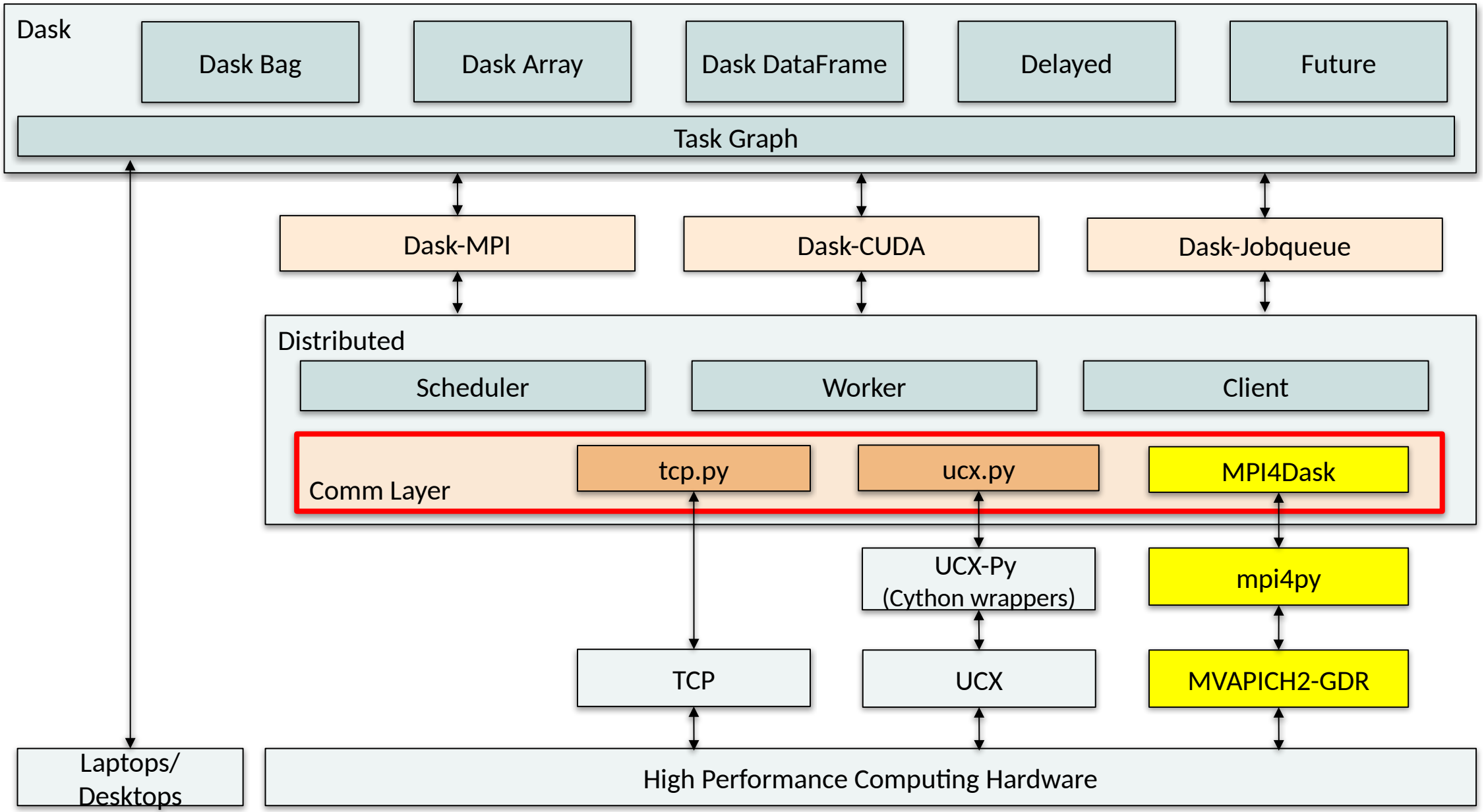
# Dask Distributed Execution Model



# MPI4Dask: MPI-based Communication Backend for Dask

- Dask originally had two communication backends:
  - TCP: Tornado-based
  - UCX: Built using a GPU-aware Cython wrapper called UCX-Py
- MPI4Dask is an MVAPICH2-based communication backend for Dask:
  - First MPI-based communication device for Dask
  - Optimizes CPU and GPU communication in the Dask framework on modern HPC clusters

# Dask Architecture



# MPI4Dask: Bootstrapping and Dynamic Connectivity

- Several ways to start Dask programs:
  - Manual
  - Utility classes:
    - LocalCUDACluster, SLURMCluster, SGECluster, PBCCluster, and others
- MPI4Dask uses the Dask-MPI to bootstrap execution of Dask programs
- Dynamic connectivity is established using the asyncio package in MPI4Dask:
  - Scheduler and workers listen for incoming connections by calling asyncio.start_server()
  - Workers and client connect using asyncio.open_connection()

# MPI4Dask: Point-to-point Communication Coroutines

- Implements communication coroutines for point-to-point MPI functions:
  - Using mpi4py (Cython wrappers) and MVAPICH2-GDR
- mpi4py provides two flavors of point-to-point communication functions:
  - Send()/Recv() – for exchanging data in buffers (faster and used in MPI4Dask)
  - send()/recv() – for communicating Python objects (pickle/unpickle)
  - GPU buffers implement the __cuda_array_interface__
- Implemented chunking mechanism for large messages
- The send and receive communication coroutines are as follows:

```
request = comm.Isend([buf, size], dest, tag)
status = request.Test()

while status is False:
   await asyncio.sleep(0)
   status = request.Test()
```

```
request = comm.Irecv([buf, size], src, tag)
status = request.Test()

while status is False:
   await asyncio.sleep(0)
   status = request.Test()
```

# MPI4Dask Installation

- MPI4Dask is available to download from: http://hibd.cse.ohio-state.edu/
  - The userguide is available at: http://hibd.cse.ohio-state.edu/static/media/hibd/dask/mpi4dask-0.2-userguide.pdf
- Section 3: Setup Instructions
  - 3.1 Installation Pre-requisites:
    - 3.1.1 Install Miniconda
    - 3.1.2 Modules/Libraries for GPU-based Dask Applications:
    - 3.1.3 Modules/Libraries for CPU-based Dask Applications:
    - 3.1.4 Install the MVAPICH2 Library (MVAPICH2-X, MVAPICH2, or MVAPICH2-GDR)
    - 3.1.5 Install the mpi4py Library
    - 3.1.6 Install Dask-MPI package
  - 3.2 Install MPI4Dask
- Section 4. Running GPU-based Dask Applications
  - 4.1 Writing the host file
  - 4.2 Sum of cuPy Array and its Transpose
  - 4.3 cuDF Merge
- Section 5. Running GPU-based Dask Applications
  - 5.1 Writing the host file
  - 5.2 Sum of numPy Array and its Transpose
  - 5.3 Sum of Pandas Data Frame
  - 5.4 SVD

# Installation Pre-requisites

## Modules/Libraries for GPU-based Dask Applications:

$ conda install -c conda-forge -c rapidsai -c nvidia automake make libtool pkg-config libhwloc psutil python=3.8 setuptools cython cudatoolkit=10.2 cupy dask-cudf dask==2021.1.1 distributed numpy rmm

## Modules/Libraries for CPU-based Dask Applications:

$ conda install -c conda-forge -c rapidsai -c nvidia automake make libtool pkg-config libhwloc psutil python=3.8 setuptools cython dask==2021.1.1 distributed=2021.1.1 numpy

## Install the MVAPICH2 Library (MVAPICH2-X, MVAPICH2, or MVAPICH2-GDR)

## Install the mpi4py Library

$ git clone https://github.com/mpi4py/mpi4py.git

$ edit mpi.cfg file

[MVAPICH2]

mpi_dir = /path/to/MVAPICH2-GDR/install/directory

mpicc = %(mpi_dir)s/bin/mpicc

mpicxx = %(mpi_dir)s/bin/mpicxx

include_dirs = %(mpi_dir)s/include

library_dirs = %(mpi_dir)s/lib64

runtime_library_dirs = %(library_dirs)s

$ python setup.py build --mpi=MVAPICH2-GDR;  $ pip install .

# Install Dask-MPI and MPI4Dask

## Install Dask-MPI package

```
$ git clone https://github.com/dask/dask-mpi.git
$ cd dask-mpi
$ python setup.py build
$ pip install .
```

## Install MPI4Dask

```
$ wget http://hibd.cse.ohio-state.edu/download/hibd/dask/mpi4dask-0.2.tar.gz
$ tar -xzvf mpi4dask-0.2.tar.gz
$ cd mpi4dask-0.2/distributed
$ python setup.py build
$ pip install .

$ conda list
$ conda list | grep distributed
$ conda list  |grep dask
```

# Running GPU and CPU Dask Applications

Sum of cuPy Array and its Transpose [GPU]

$ cd mpi4dask-0.2/dask-apps/gpu

$ LD_PRELOAD=$MPILIB/lib64/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1 MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python cupy_sum_mpi.py

cuDF Merge [GPU]

$ cd mpi4dask-0.2/dask-apps/gpu

$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile hosts MV2_USE_CUDA=1 MV2_USE_GDRCOPY=1 MV2_GPUDIRECT_GDRCOPY_LIB=/opt/gdrcopy2.0/lib64/libgdrapi.so MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python cudf_merge_mpi.py --type gpu --protocol mpi --runs 20 --chunk-size 1_000_000_00

Sum of numPy Array and its Transpose [CPU]

$ cd mpi4dask-0.2/dask-apps/cpu

$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile hosts MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python numpy_sum_mpi.py

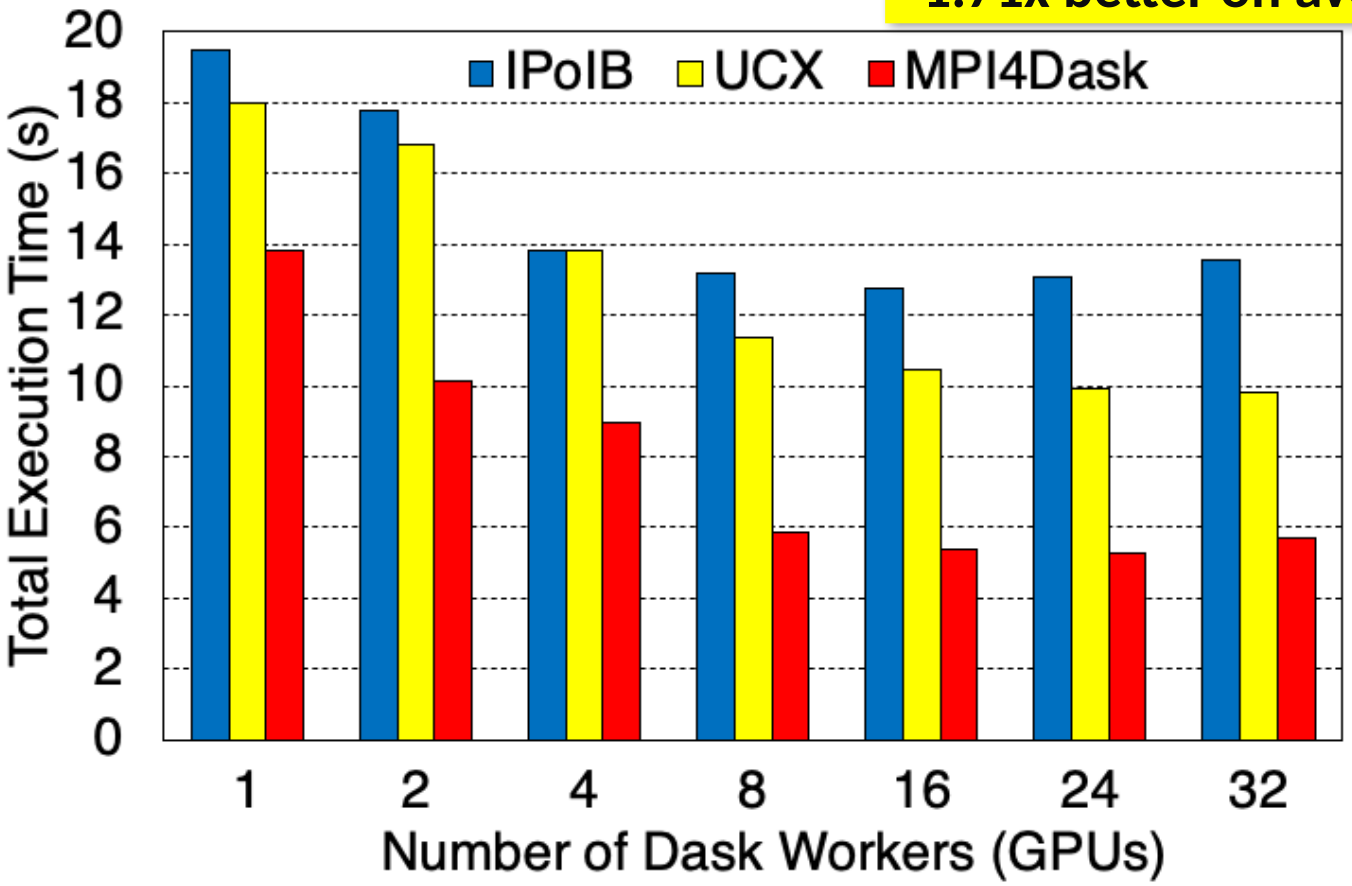
Sum of Pandas DataFrame [CPU]

$ cd mpi4dask-0.2/dask-apps/cpu

$ LD_PRELOAD=$MPILIB/lib/libmpi.so $MPILIB/bin/mpirun_rsh -export-all -np 4 -hostfile hosts MV2_CPU_BINDING_LEVEL=SOCKET MV2_CPU_BINDING_POLICY=SCATTER python dask-cudf_sum_mpi.py

# Benchmark: Sum of cuPy Array and its Transpose (TACC Frontera GPU Subsystem)

1.71x better on average




A. Shafi , J. Hashmi , H. Subramoni , and D. K. Panda, Efficient MPI-based Communication for GPU-Accelerated Dask Applications, CCGrid '21
https://arxiv.org/abs/2101.08878

MPI4Dask 0.2 release

(http://hibd.cse.ohio-state.edu)

# Outline

- Introduction
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- Deep Learning
  - Deep Neural Networks
  - Distributed Deep Learning
  - DL Solutions
- Open Issues and Challenges
- **Conclusion**

# Conclusion

- Exponential growth in Machine Learning/Deep Learning/Data Science frameworks
- Provided an overview of issues, challenges, and opportunities for designing efficient communication runtimes
  - Efficient, scalable, and hierarchical designs are crucial for ML/DL/Data Science frameworks
  - Co-design of communication runtimes and ML/DL/Data Science frameworks will be essential
- Presented use-cases to demonstrate the complex interaction between DL/ML/Dask middleware with the underling HPC technologies and middleware
- Need collaborative efforts to achieve the full potential

# Funding Acknowledgments

## *Funding Support by*


## *Equipment Support by*

# Acknowledgments to all the Heroes (Past/Current Students and Staffs)


***Current Students (Graduate)***

- N. Alnaasan (Ph.D.)
- Q. Anthony (Ph.D.)
- C.-C. Chun (Ph.D.)
- N. Contini (Ph.D.)
- A. Jain (Ph.D.)
- K. S. Khorassani (Ph.D.)
- P. Kousha (Ph.D.)
- B. Michalowicz (Ph.D.)
- B. Ramesh (Ph.D.)
- K. K. Suresh (Ph.D.)
- A. H. Tu (Ph.D.)
- S. Xu (Ph.D.)
- Q. Zhou (Ph.D.)
- K. Al Attar (M.S.)
- L. Xu (Ph.D.)
- H. Ahn (Ph.D.)
- G. Kuncham (Ph.D.)
- R. Vaidya (Ph.D.)
- J. Yao (P.hD.)
- M. Han (M.S.)
- A. Guptha (M.S.)

***Current Research Scientists***

- M. Abduljabbar
- A. Shafi

***Current Students (Undergrads)***

- V. Shah
- T. Chen

***Current Faculty***

- H. Subramoni

***Current Software Engineers***

- B. Seeds
- N. Pavuk
- N. Shineman
- M. Lieber

***Current Research Specialist***

- R. Motlagh

***Past Students***

- A. Awan (Ph.D.)
- A. Augustine (M.S.)
- P. Balaji (Ph.D.)
- M. Bayatpour (Ph.D.)
- R. Biswas (M.S.)
- S. Bhagvat (M.S.)
- A. Bhat (M.S.)
- D. Buntinas (Ph.D.)
- L. Chai (Ph.D.)
- B. Chandrasekharan (M.S.)
- S. Chakraborthy (Ph.D.)
- N. Dandapanthula (M.S.)
- V. Dhanraj (M.S.)
- C.-H. Chu (Ph.D.)
- T. Gangadharappa (M.S.)
- K. Gopalakrishnan (M.S.)
- J. Hashmi (Ph.D.)
- W. Huang (Ph.D.)
- W. Jiang (M.S.)
- J. Jose (Ph.D.)
- M. Kedia (M.S.)
- S. Kini (M.S.)
- M. Koop (Ph.D.)
- K. Kulkarni (M.S.)
- R. Kumar (M.S.)
- S. Krishnamoorthy (M.S.)
- K. Kandalla (Ph.D.)
- M. Li (Ph.D.)
- P. Lai (M.S.)
- J. Liu (Ph.D.)
- M. Luo (Ph.D.)
- A. Mamidala (Ph.D.)
- G. Marsh (M.S.)
- V. Meshram (M.S.)
- A. Moody (M.S.)
- S. Naravula (Ph.D.)
- R. Noronha (Ph.D.)
- X. Ouyang (Ph.D.)
- S. Pai (M.S.)
- S. Potluri (Ph.D.)
- K. Raj (M.S.)
- R. Rajachandrasekar (Ph.D.)
- D. Shankar (Ph.D.)
- G. Santhanaraman (Ph.D.)
- N. Sarkauskas (B.S. and M.S)
- N. Senthil Kumar (M.S.)
- A. Singh (Ph.D.)
- J. Sridhar (M.S.)
- S. Srivastava (M.S.)
- S. Sur (Ph.D.)
- H. Subramoni (Ph.D.)
- K. Vaidyanathan (Ph.D.)
- A. Vishnu (Ph.D.)
- J. Wu (Ph.D.)
- W. Yu (Ph.D.)
- J. Zhang (Ph.D.)

***Past Post-Docs***

- D. Banerjee
- X. Besseron
- M. S. Ghazimeersaeed
- H.-W. Jin
- J. Lin
- M. Luo
- E. Mancini
- K. Manian
- S. Marcarelli
- A. Ruhela
- J. Vienne
- H. Wang

***Past Research Scientists***

- K. Hamidouche
- S. Sur
- X. Lu

***Past Senior Research Associate***

- J. Hashmi

***Past Programmers***

- A. Reifsteck
- D. Bureddy
- J. Perkins

***Past Research Specialist***

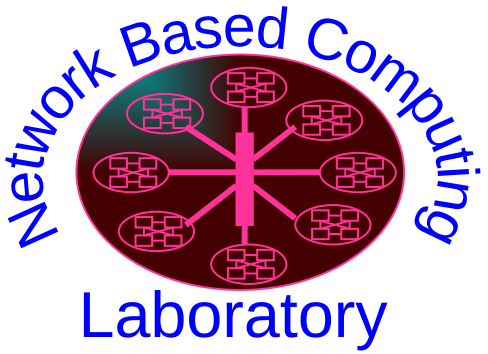
- M. Arnold
- J. Smith

# Thank You!

{shafi.16, jain.575}@osu.edu

*Follow us on*

https://twitter.com/mvapich




Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/


The MVAPICH2 Project
http://mvapich.cse.ohio-state.edu/


The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/