MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Follow us on

https://twitter.com/mvapich

HiDL
High-Performance Deep Learning

High Performance Machine Learning and Deep Learning with MVAPICH2

Tutorial at MUG '23

by

Aamir Shafi

The Ohio State University

shafi.16@osu.edu

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

Nawras Alnaasan

The Ohio State University

alnaasan.1@osu.edu

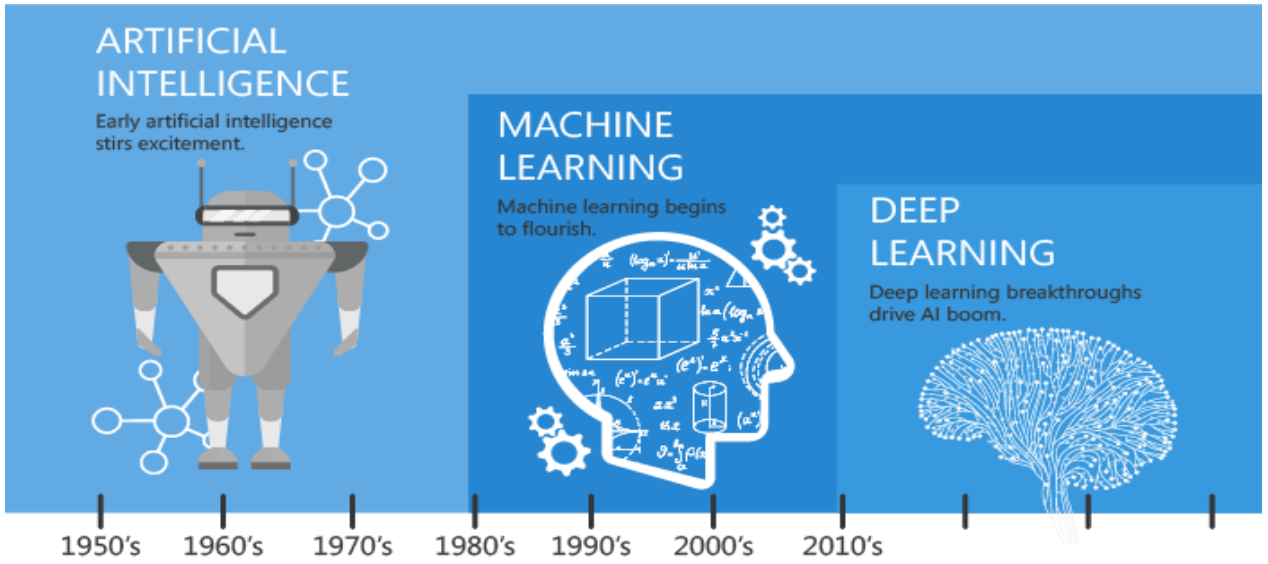
https://engineering.osu.edu/people/alnaasan.1

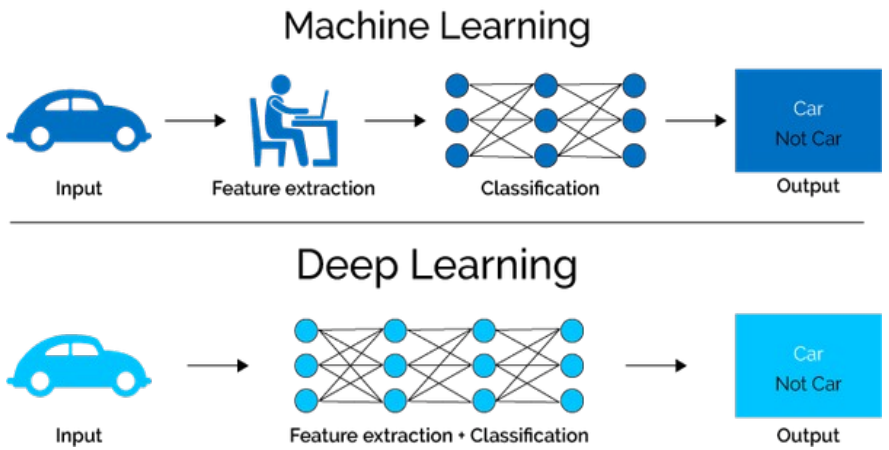
Outline

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

What is Machine Learning and Deep Learning?

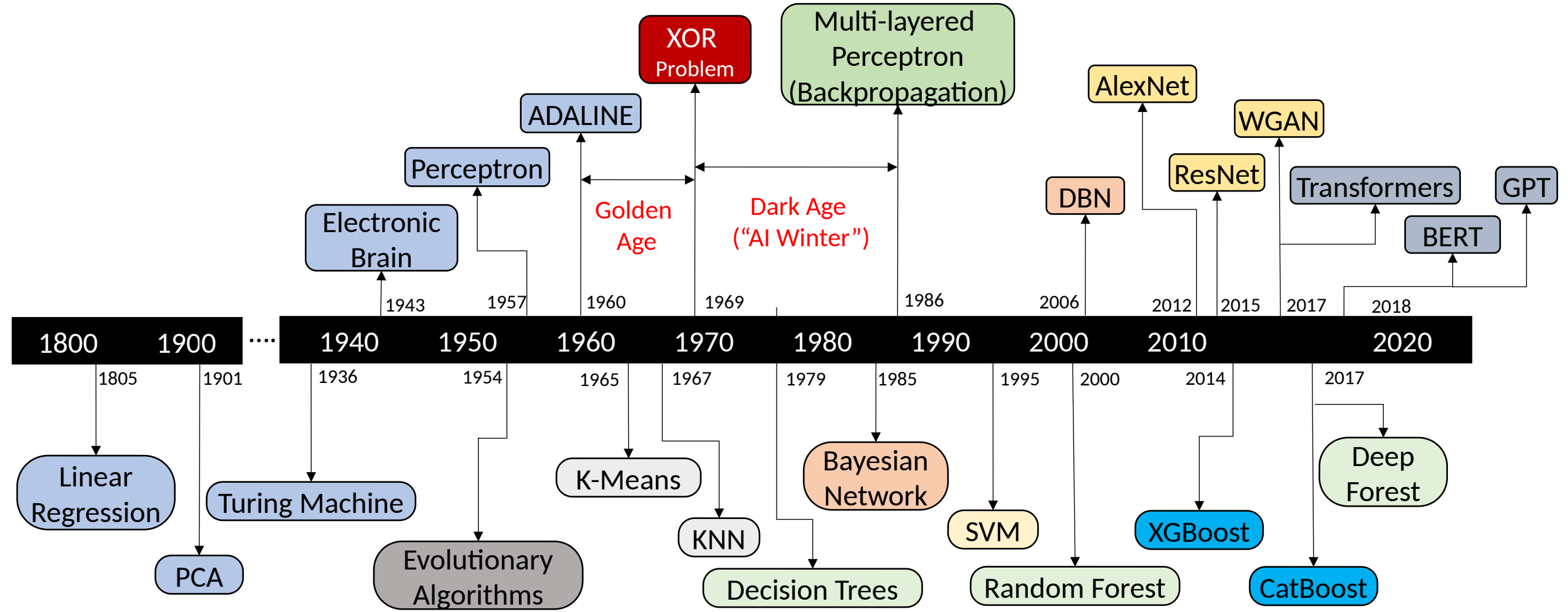
- Machine Learning (ML)
  - "the study of computer algorithms to improve automatically through experience and use of data"
- Deep Learning (DL) – a subset of ML
  - Uses Deep Neural Networks (DNNs)
  - **Perhaps, the most revolutionary subset!**
- Based on learning data representation
- DNN Examples: 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

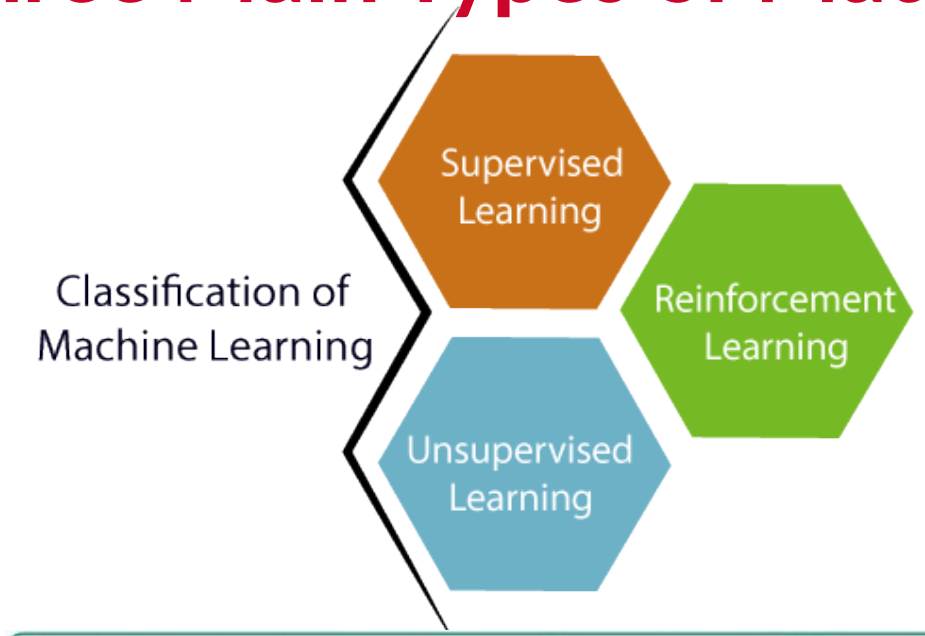


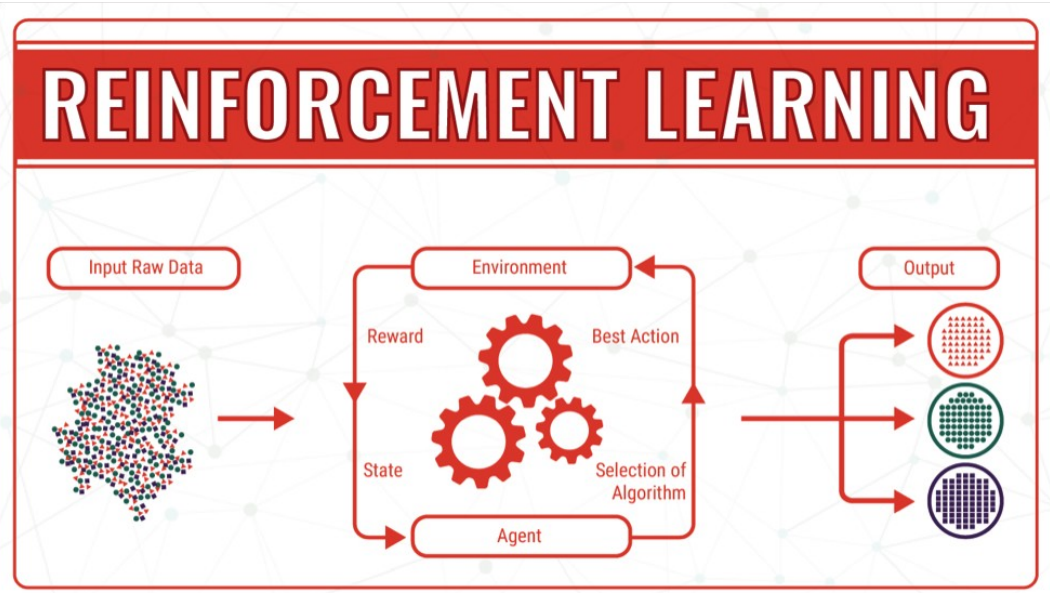
A. Legendre – J. Gauss | K. Pearson | A. Turing | S. McCulloch – W. Pitts | F. Rosenblatt | B. Widrow – M. Hoff | M. Minsky – S. Papert | J. Pearl | D. Rumelhart – G. Hinton – R. Wiliams | V. Vapnik– C. Cortes | A. Krizhevsky | A. Ng | Y. LeCun | Y. Bengio

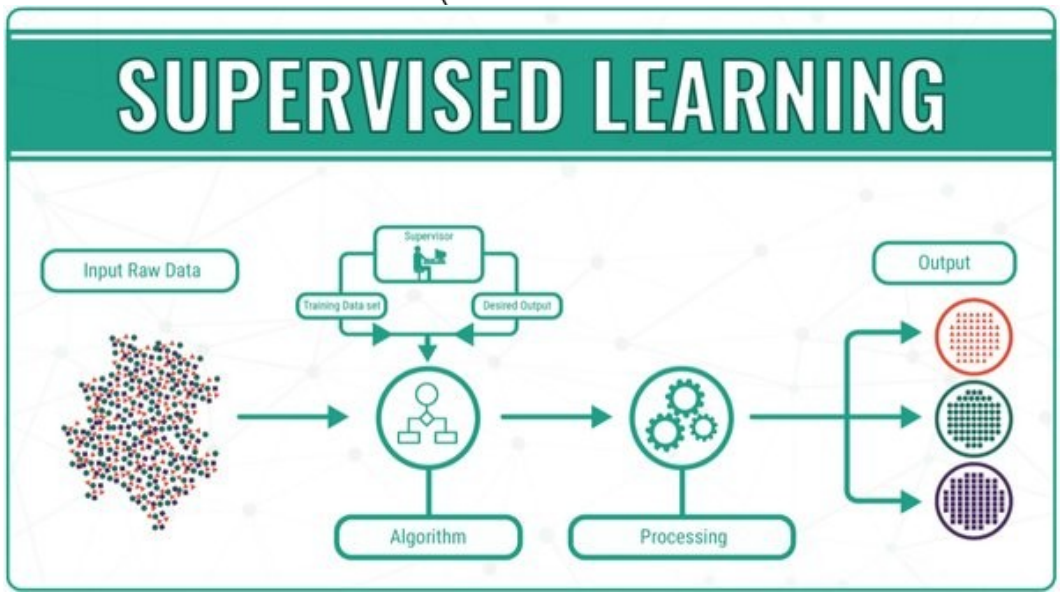
Outline

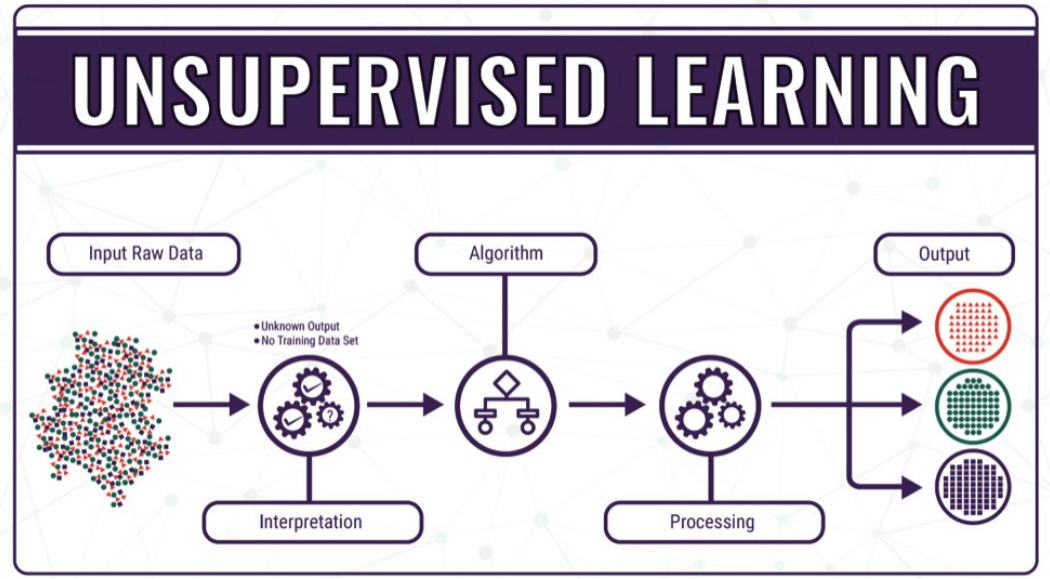
- Introduction
- **Machine Learning**
  - **Distributed K-Means**
  - ML Solutions
- Deep Learning
  - Deep Neural Networks
  - Distributed Deep Learning
  - DL Solutions
- Conclusion

Three Main Types of Machine Learning









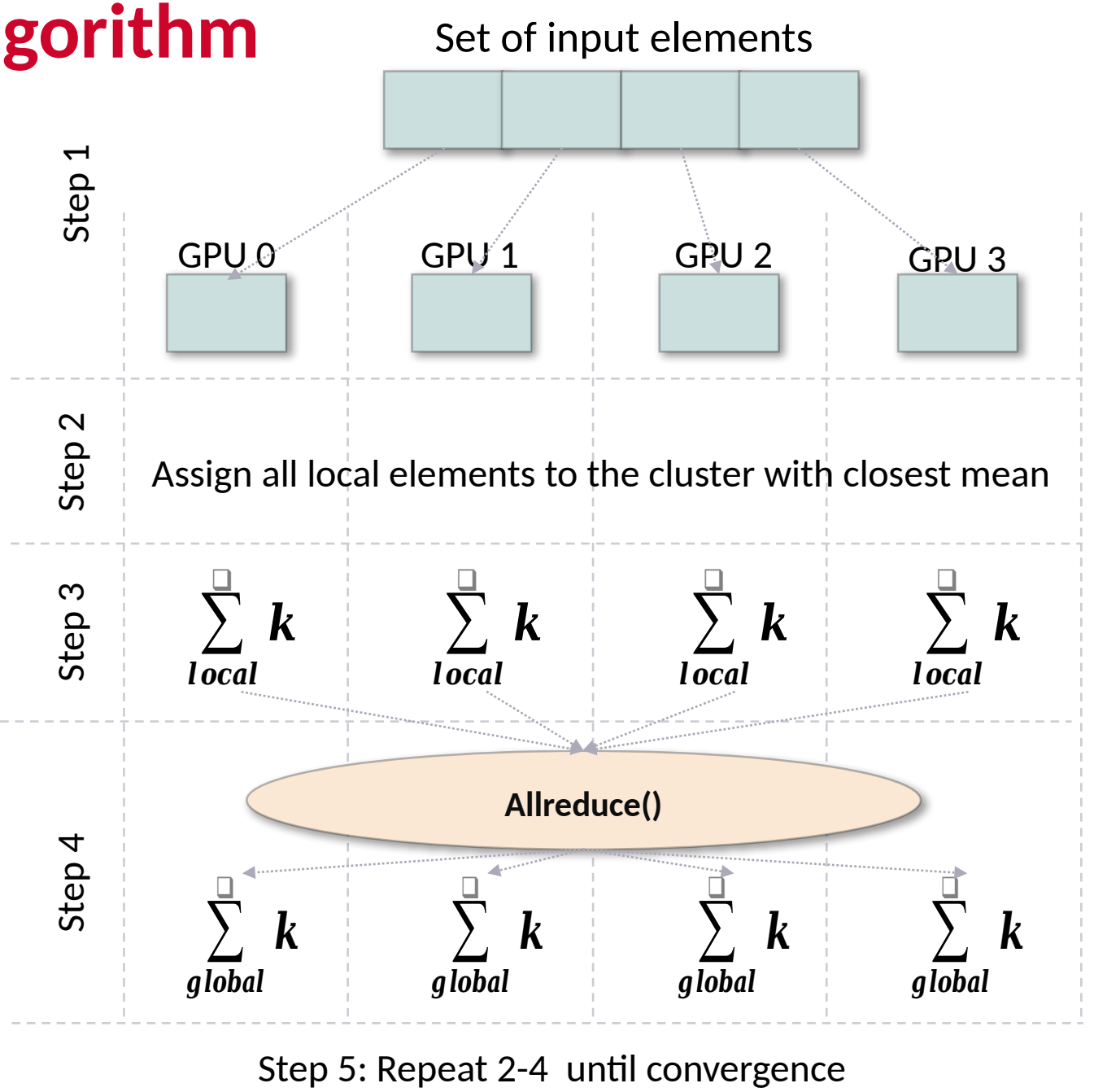
Courtesy: https://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/

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

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

Outline

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

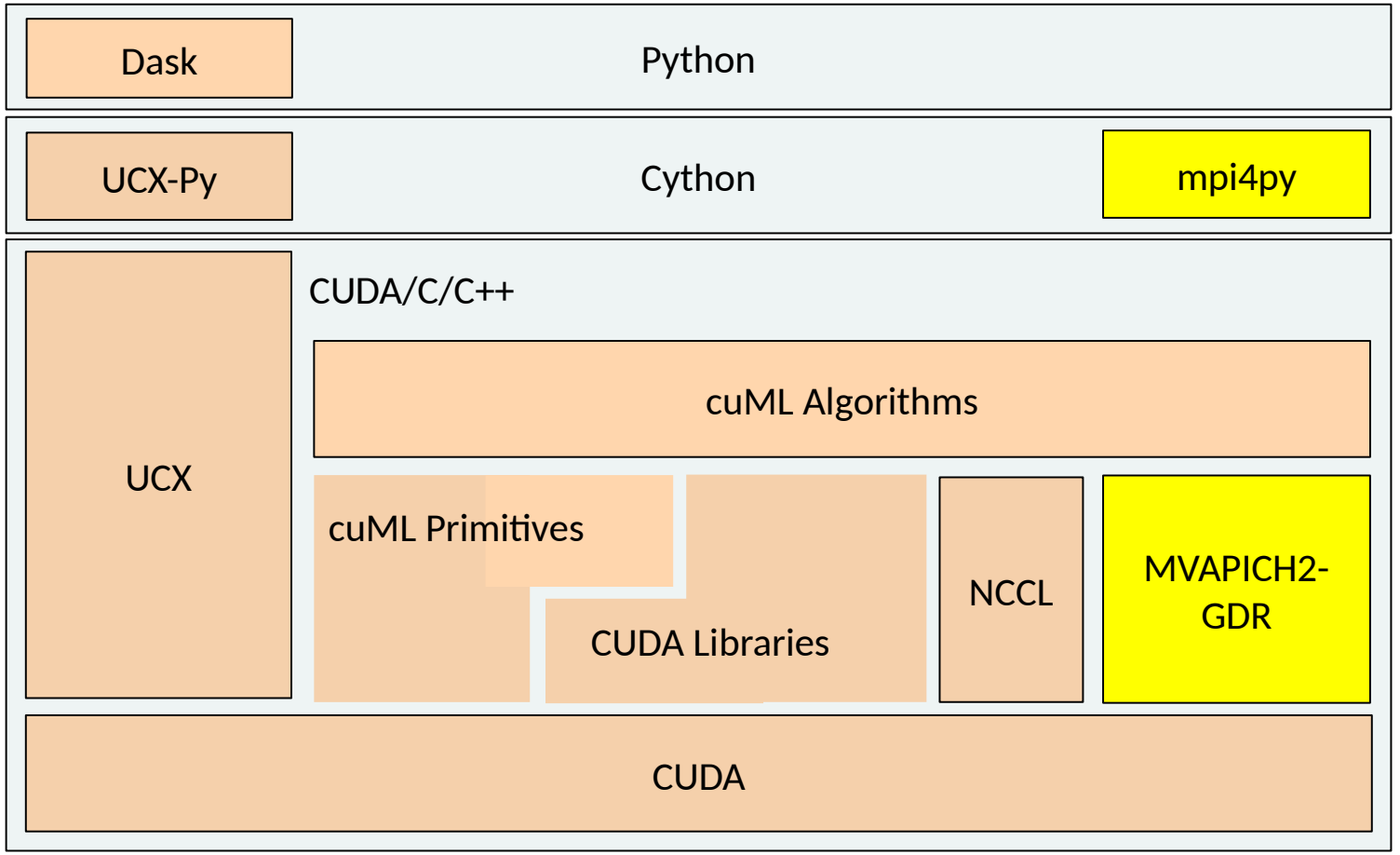
- Main components
  - Python layer
    - Provides a Scikit-learn like interface
    - Hides the complexities of the CUDA/C/C++ layer
  - Primitives and cuML algorithms built on top of CUDA
    - ML Algorithms
    - Primitives
      - Reusable building blocks for building machine learning algorithms
      - 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 0.5 release

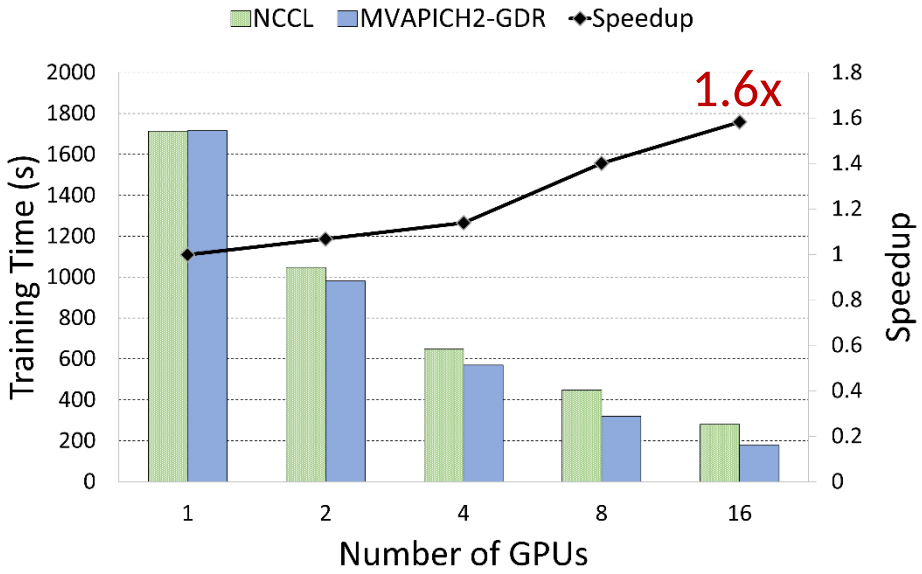
(http://hidl.cse.ohio-state.edu)



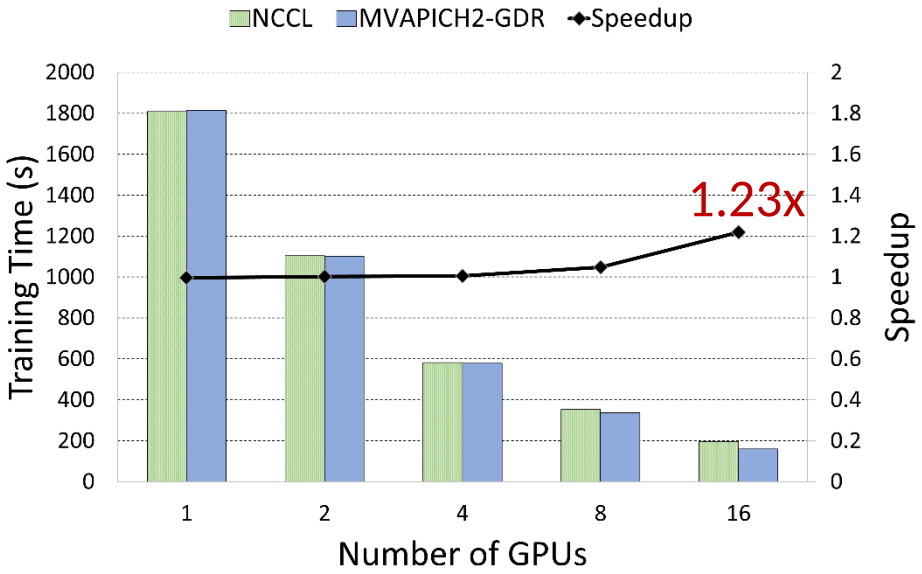
MPI4cuML 0.5 Release - MPI-Driven ML Training

- cuML is a distributed machine learning training framework with a focus on GPU acceleration and distributed computing. MVAPICH2-GDR provides many features to augment distributed training with cuML on GPUs
- (NEW) Based on cuML 22.02.00
  - Include ready-to-use examples for KMeans, Linear Regression, Nearest Neighbors, and tSVD
- (NEW) MVAPICH2 support for RAFT 22.02.00
  - (NEW) Enabled cuML's communication engine, RAFT, to use MVAPICH2-GDR backend for Python and C++ cuML applications
  - KMeans, PCA, tSVD, RF, LinearModels
  - Added switch between available communication backends (MVAPICH2 and NCCL)
- Built on top of mpi4py over the MVAPICH2-GDR library
- Tested with
  - Mellanox InfiniBand adapters (FDR and HDR)
  - (NEW) NVIDIA GPU A100, V100 and, P100
  - Various x86-based multi-core platforms (AMD and Intel)
- http://hidl.cse.ohio-state.edu/

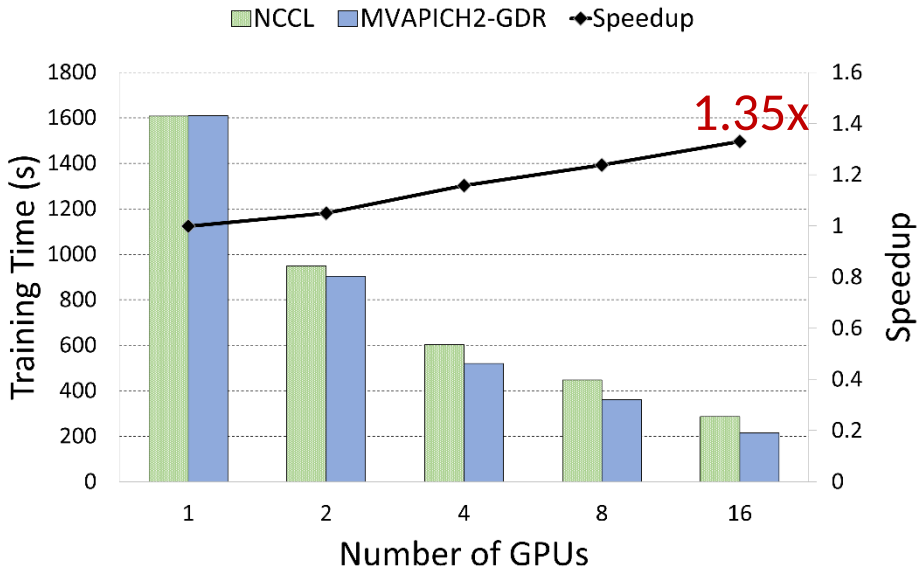
K-Means



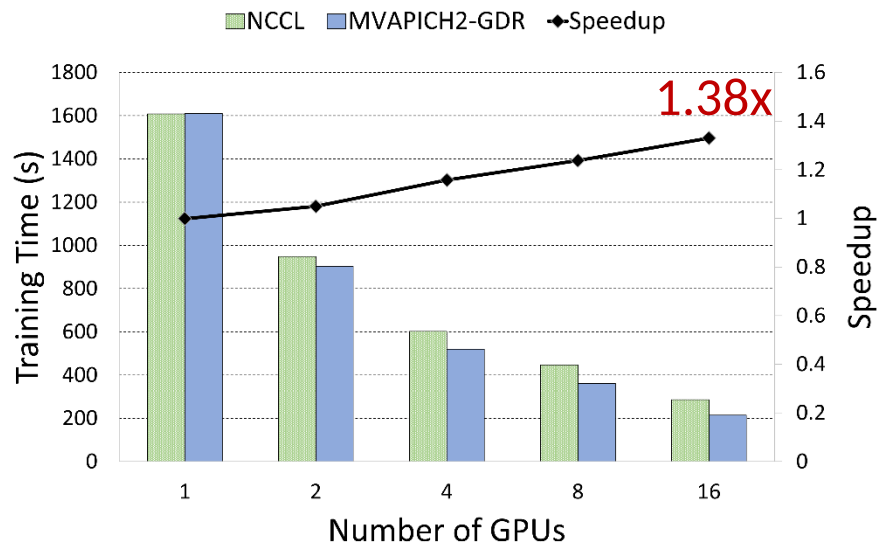
Linear Regression



Nearest Neighbors



Truncated SVD



MPI4cuML 0.5

Expanse GPU System	
CPU Model	Intel Xeon Gold 6248
CPU Core Info	2x20 @ 2.5Ghz
Memory	384 GB
Interconnect	Infiniband HDR (200 Gbps)
OS	Rocky Linux 8.5
GPU	NVIDIA V100 (4/Node)
CUDA	CUDA 11.2

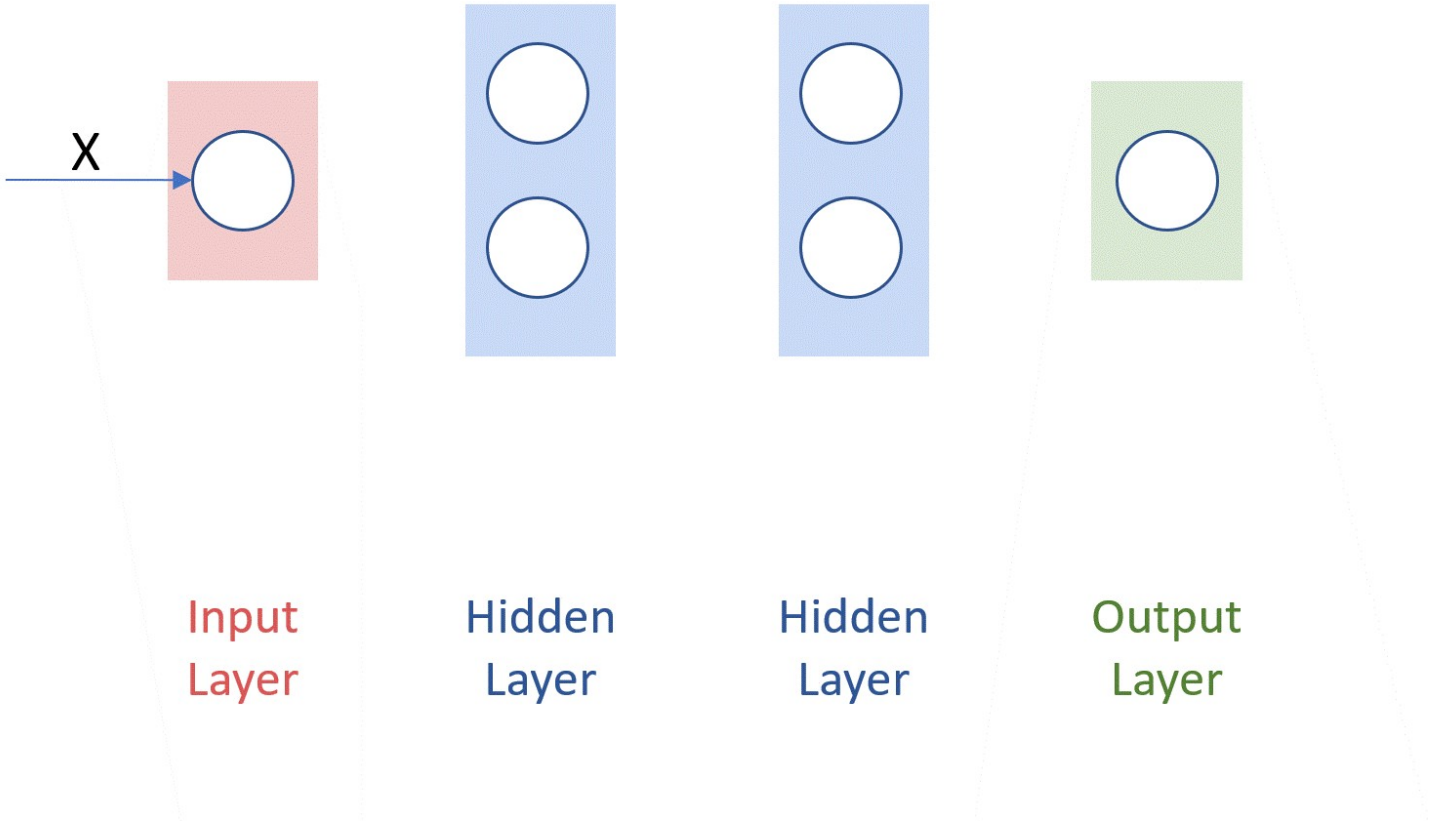
M. Ghazimirsaeed , Q. Anthony , A. Shafi , H. Subramoni , and D. K. Panda, Accelerating GPU-based Machine Learning in Python using MPI Library: A Case Study with MVAPICH2-GDR, MLHPC Workshop, Nov 2020

Outline

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

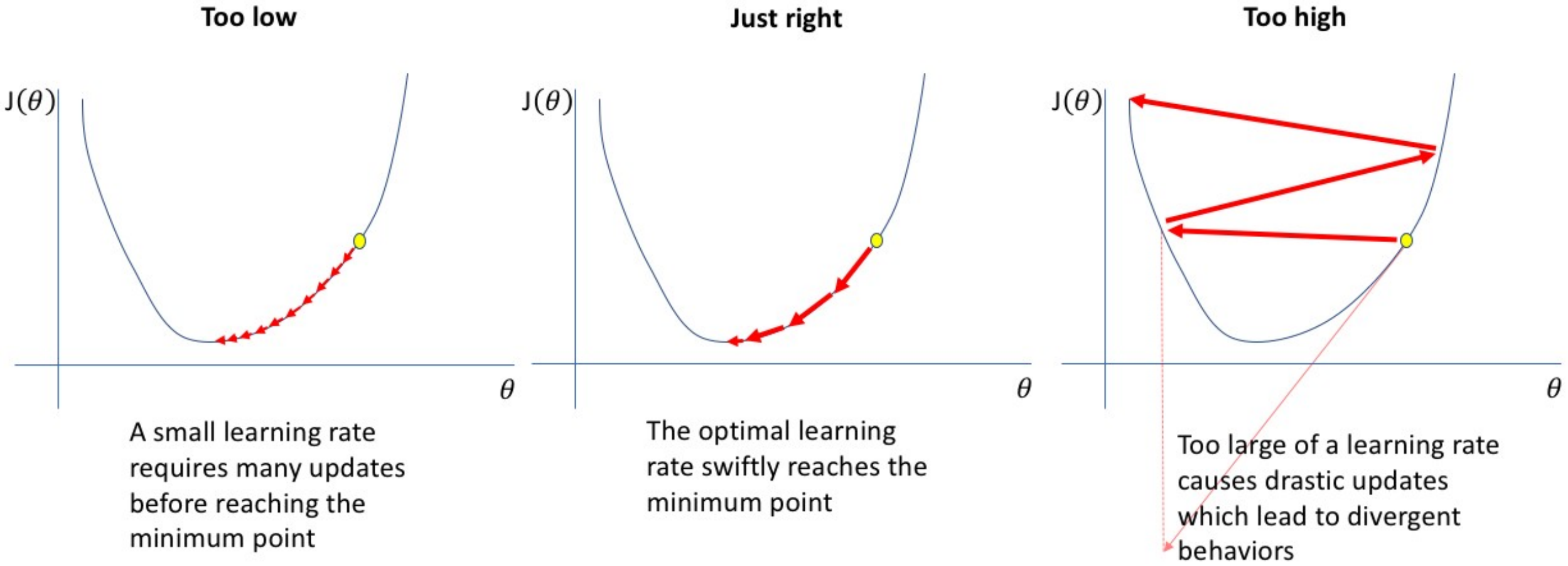
Understanding the Deep Neural Network Concepts

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



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

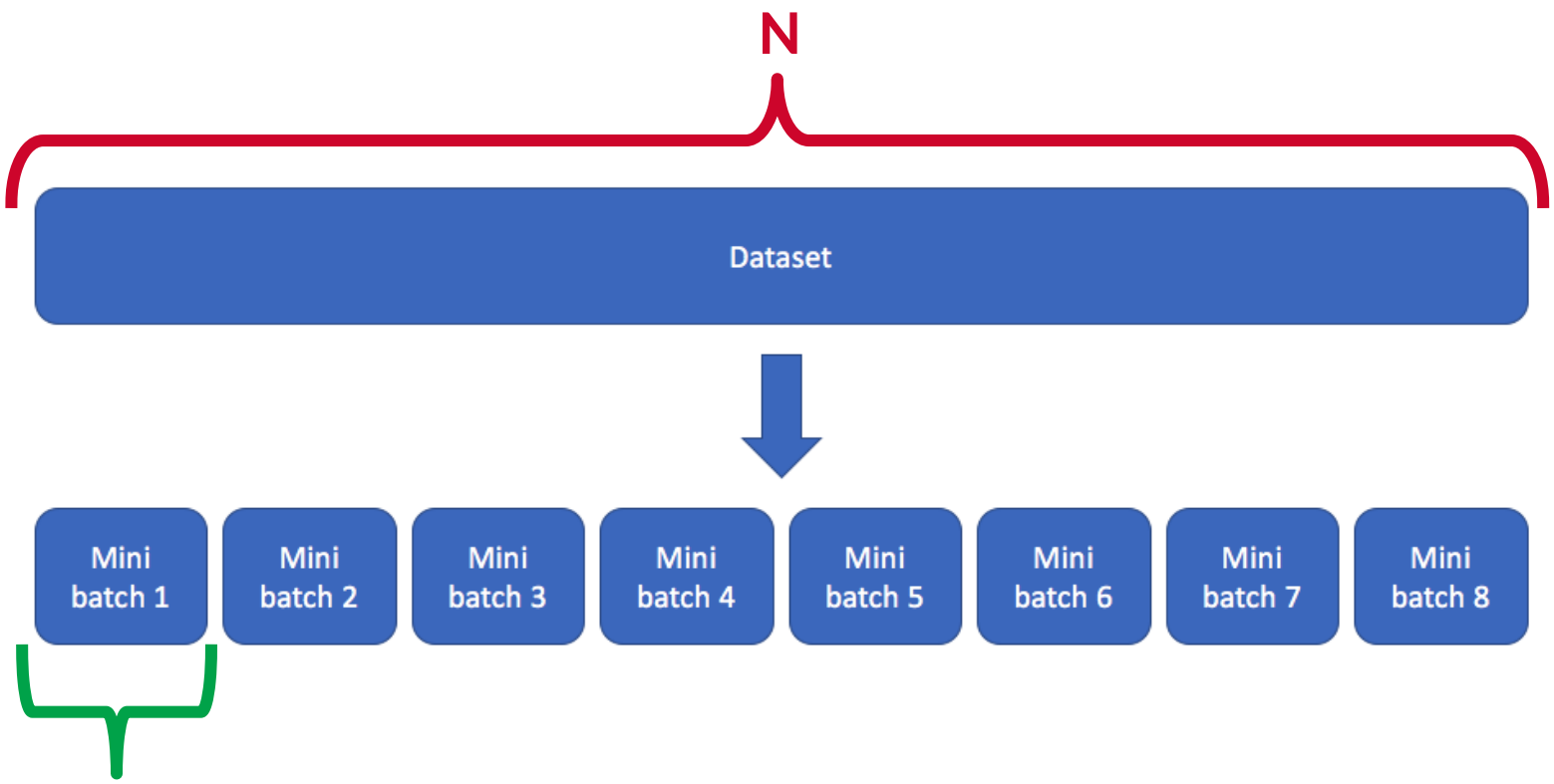
Essential Concepts: Learning Rate (α)



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

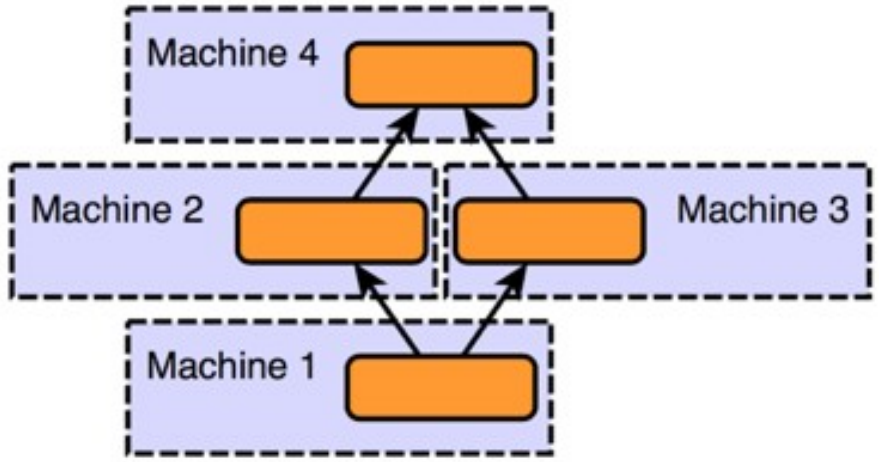
- Introduction
- Machine Learning
  - Distributed K-Means
  - ML Solutions
- **Deep Learning**
  - Deep Neural Networks
  - **Distributed Deep Learning**
  - DL Solutions
- Conclusion

The Need for Parallel and Distributed Training

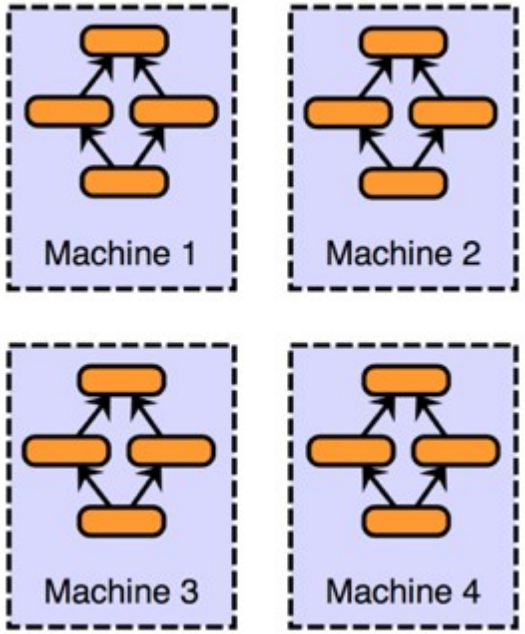
- Why do we need Parallel Training?
- Larger and Deeper models are being proposed
  - **AlexNet -> ResNet -> NASNet – AmoebaNet**
  - DNNs require a lot of memory and a lot of computation
  - Larger models cannot fit a GPU's memory
- Single GPU training cannot keep up with ever-larger models
- Community 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

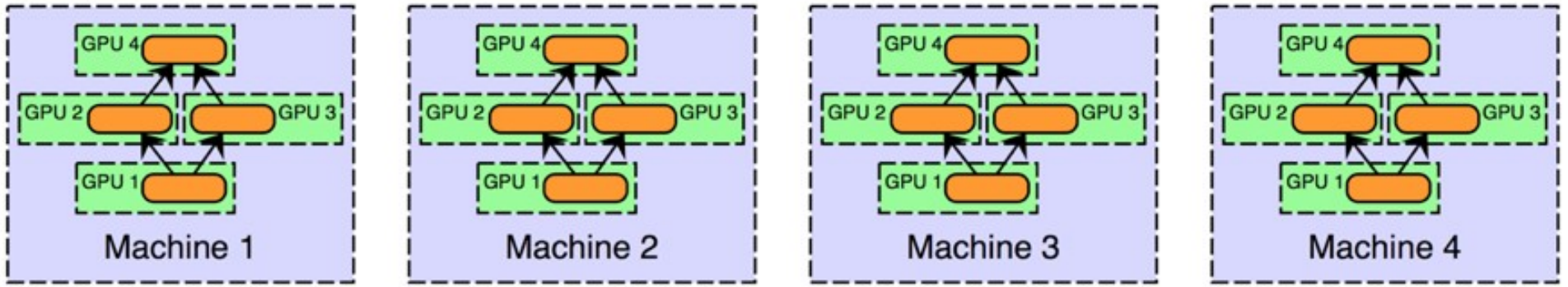
- Some parallelization strategies..
  - Data Parallelism or Model Parallelism
  - Hybrid Parallelism



Model Parallelism



Data Parallelism

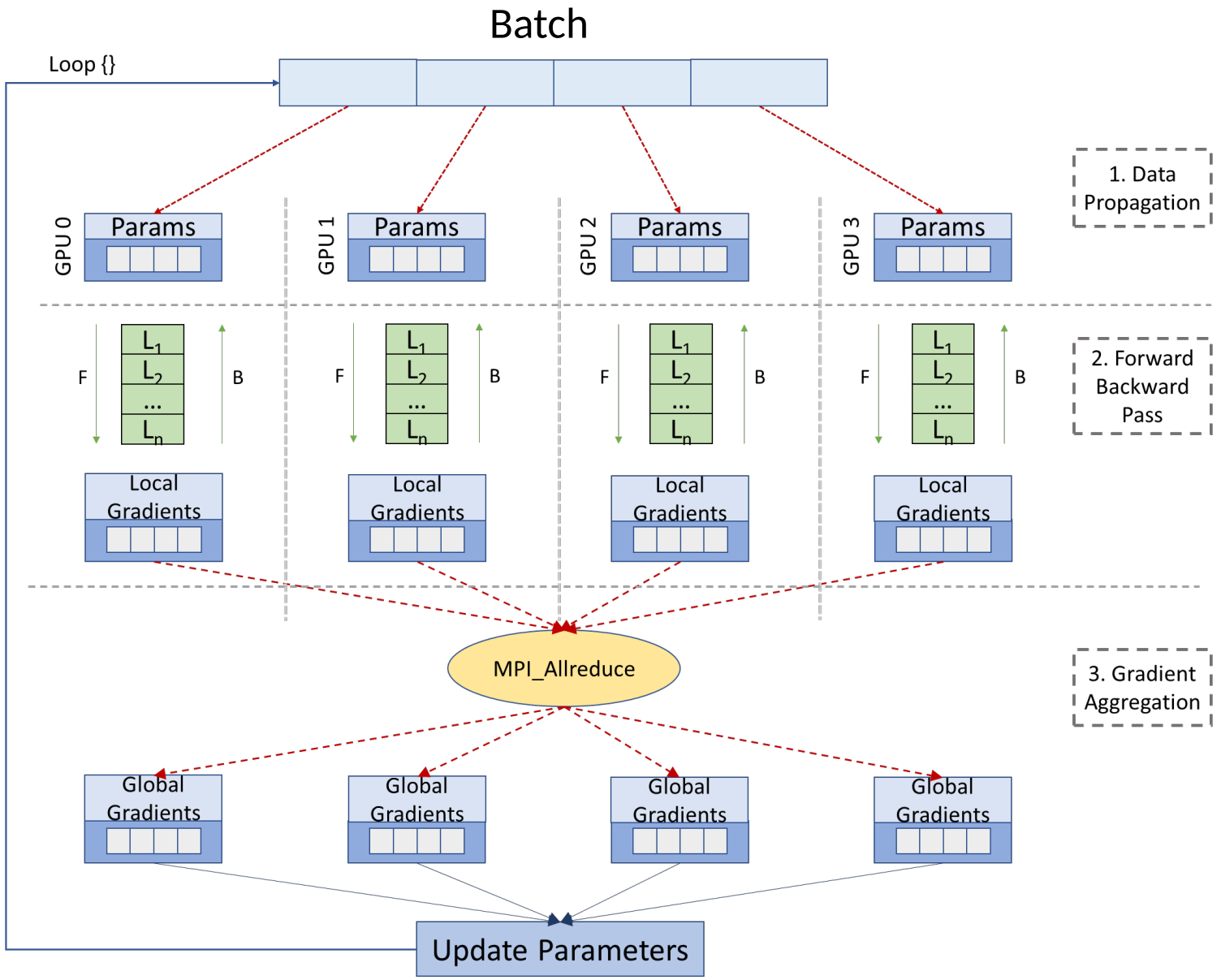


Hybrid (Model and Data) Parallelism

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

Data Parallelism and MPI Collectives

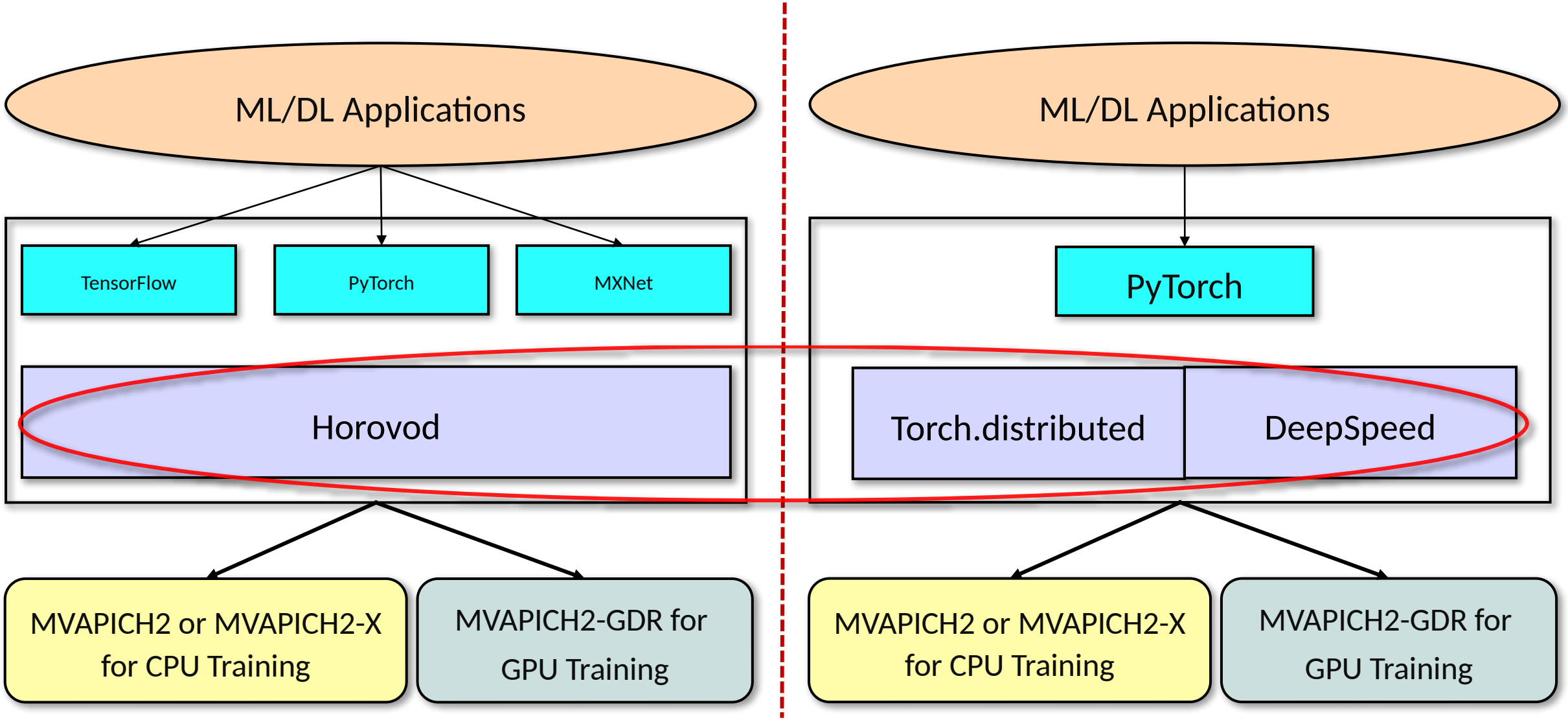
- Step1: Data Propagation
  - Distribute the Data among GPUs
- Step2: Forward Backward Pass
  - Perform forward pass and calculate the prediction
  - Calculate Error by comparing prediction with actual output
  - Perform backward pass and calculate gradients
- Step3: Gradient Aggregation
  - Call MPI_Allreduce 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**
- Conclusion

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 for CPU

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

Command to install Horovod for GPU

```
$ 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

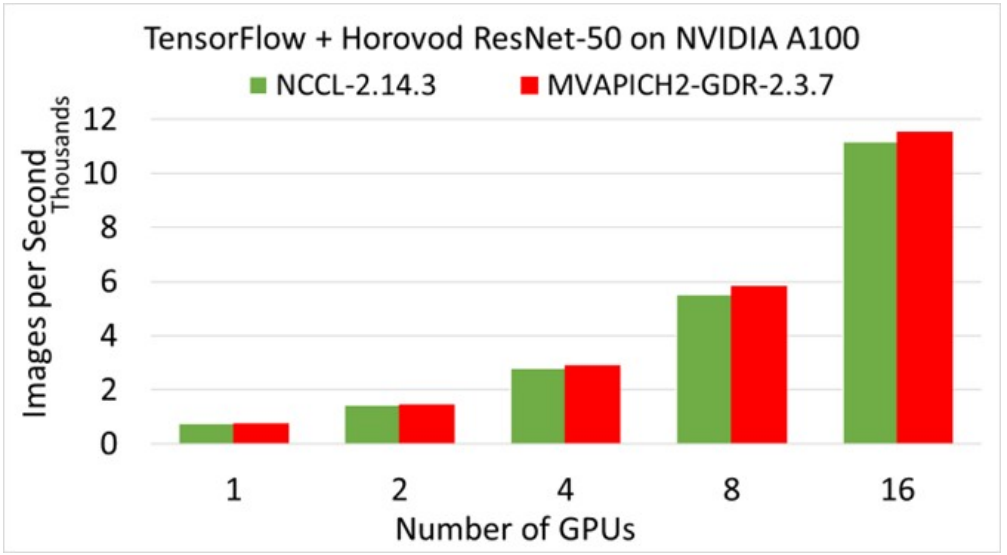
V100

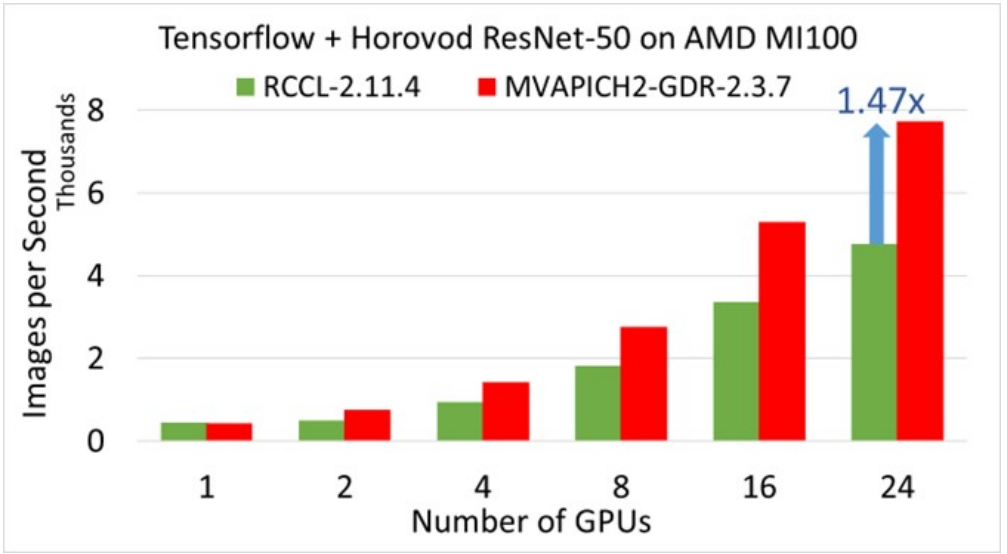
```
.
.
Model: resnet50
Batch size: 64
Number of GPUs: 2
Running warmup...
Running benchmark...
Iter #0: 317.0 img/sec per GPU
Iter #1: 314.9 img/sec per GPU
Iter #2: 315.4 img/sec per GPU
Iter #3: 318.0 img/sec per GPU
Iter #4: 316.7 img/sec per GPU
Img/sec per GPU: 316.4 +-2.2
-----------------------------------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

~1.89X on 2 GPUs

HiDL Software Stack Release v1.0

- Based on Horovod
- Optimized support for MPI controller in deep learning workloads
- Efficient large-message collectives (e.g. Allreduce) on various CPUs and GPUs
- GPU-Direct algorithms for collective operations (including those commonly used for data- and model-parallelism, e.g. Allgather and Alltoall)
- Support for fork safety
- Exploits efficient large message collectives
- Compatible with
  - Mellanox InfiniBand adapters (EDR, FDR, HDR)
  - Various x86-based multi-core CPUs (AMD and Intel)
  - NVIDIA A100, V100, P100, Quadro RTX 5000 GPUs
  - CUDA [9.x, 10.x, 11.x] and cuDNN [7.5.x, 7.6.x, 8.0.x, 8.2.x, 8.4.x]
  - AMD MI100 GPUs
  - ROCm [5.1.x]

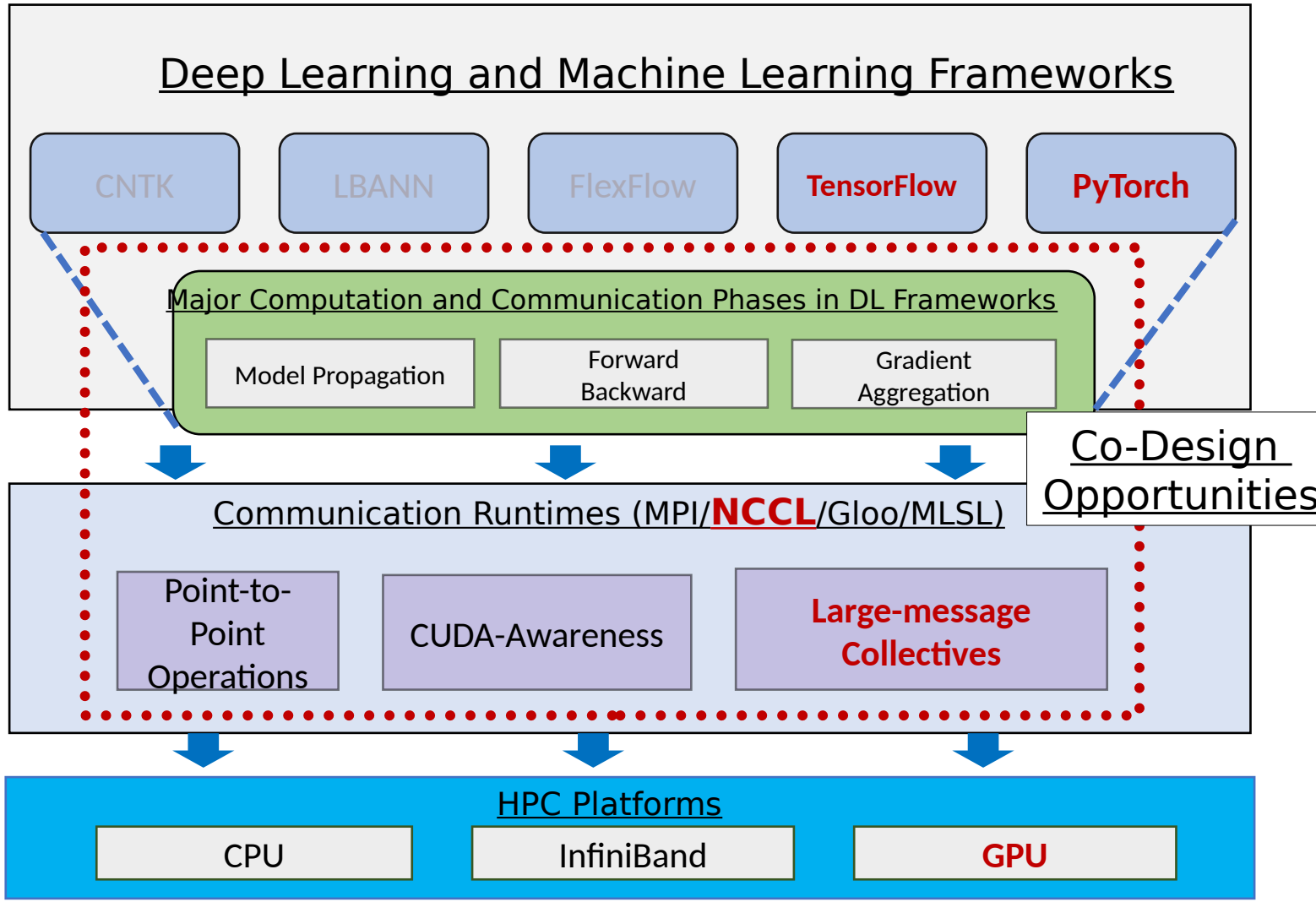




For more details: http://hidl.cse.ohio-state.edu/userguide/horovod/

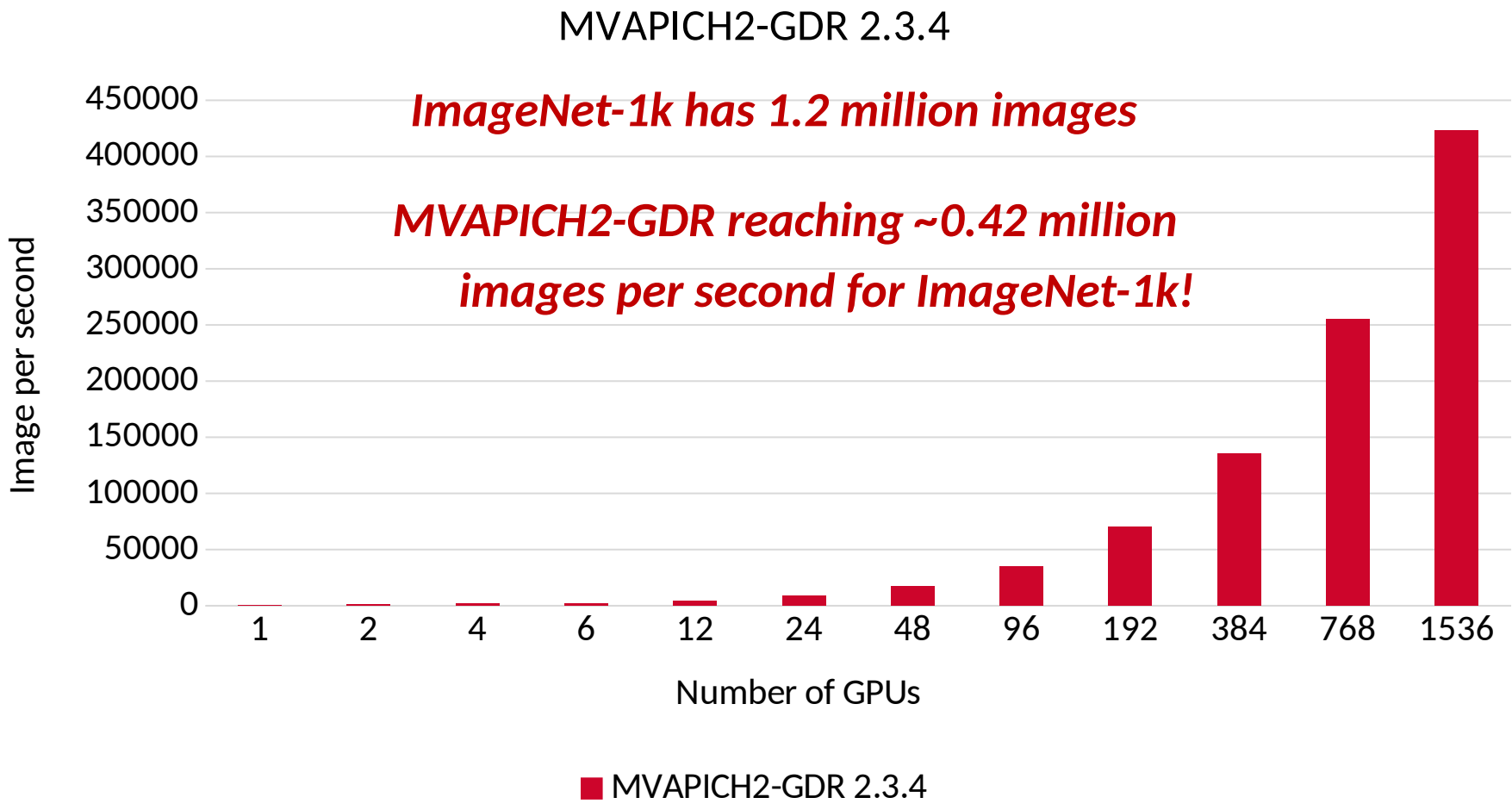
Solutions and Case Studies: Exploiting HPC for DL

- **Data Parallelism**
- Model-Parallelism



Distributed TensorFlow on ORNL Summit (1,536 GPUs)

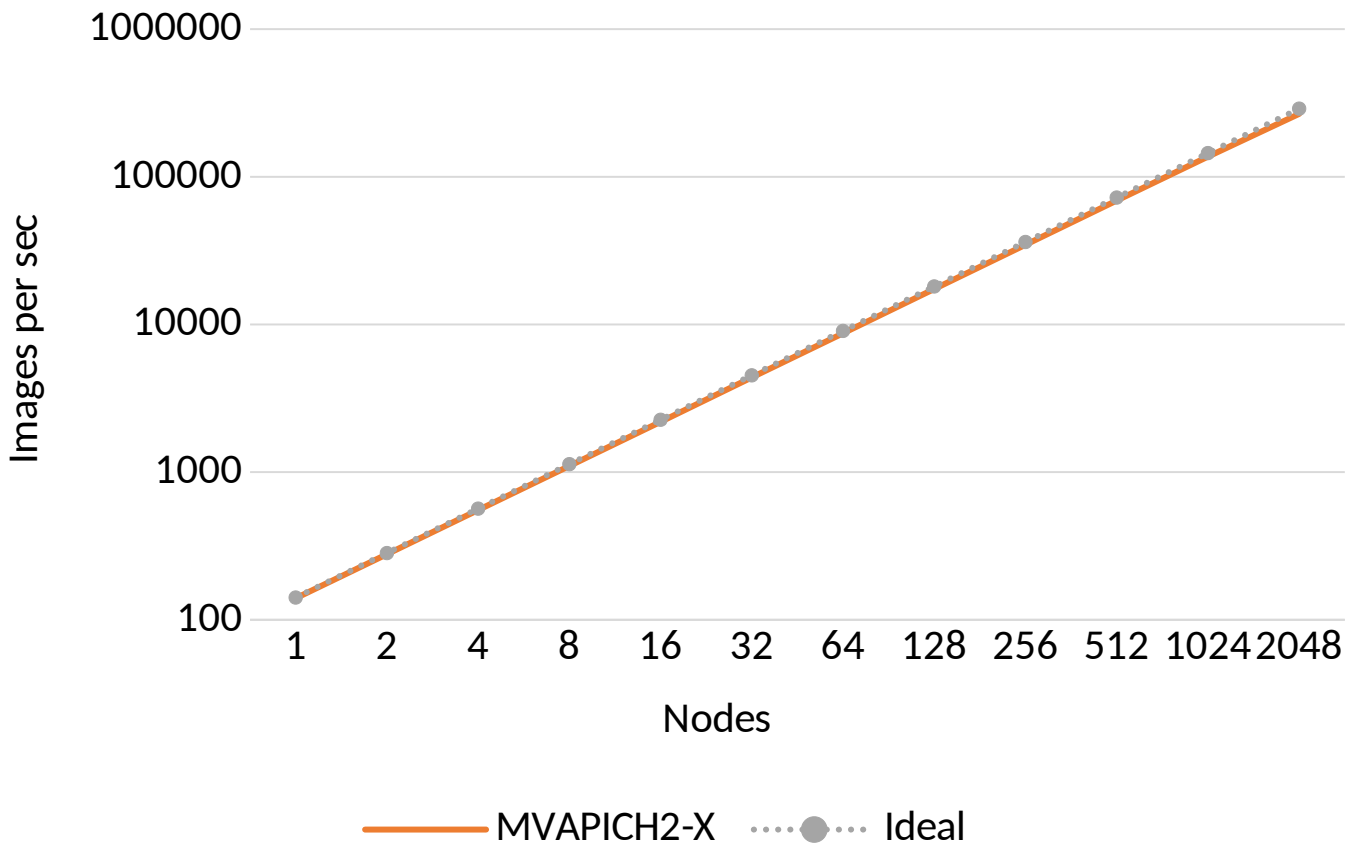
- ResNet-50 Training using TensorFlow benchmark on SUMMIT -- 1536 Volta GPUs!
- 1,281,167 (1.2 mil.) images
- Time/epoch = 3 seconds
- Total Time (90 epochs) = 3 x 90 = 270 seconds = **4.5 minutes!**



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!

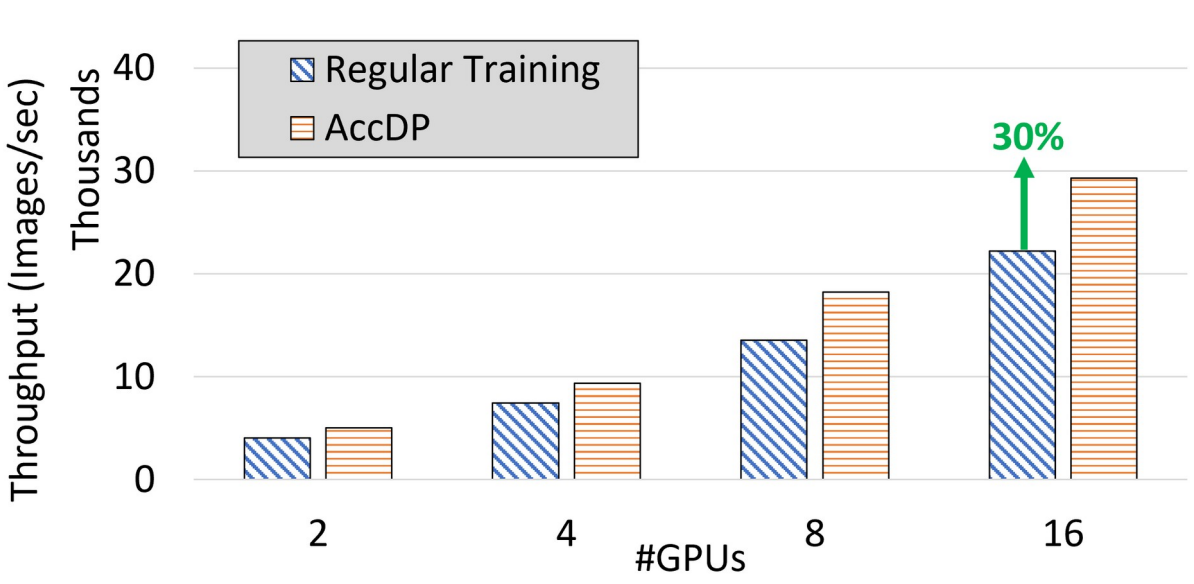


A. Jain, A. A. Awan, H. Subramoni, DK Panda, "Scaling TensorFlow, PyTorch, and MXNet using MVAPICH2 for High-Performance Deep Learning on Frontera", DLS '19 (SC '19 Workshop).

AccDP: Exploiting Data Parallelism

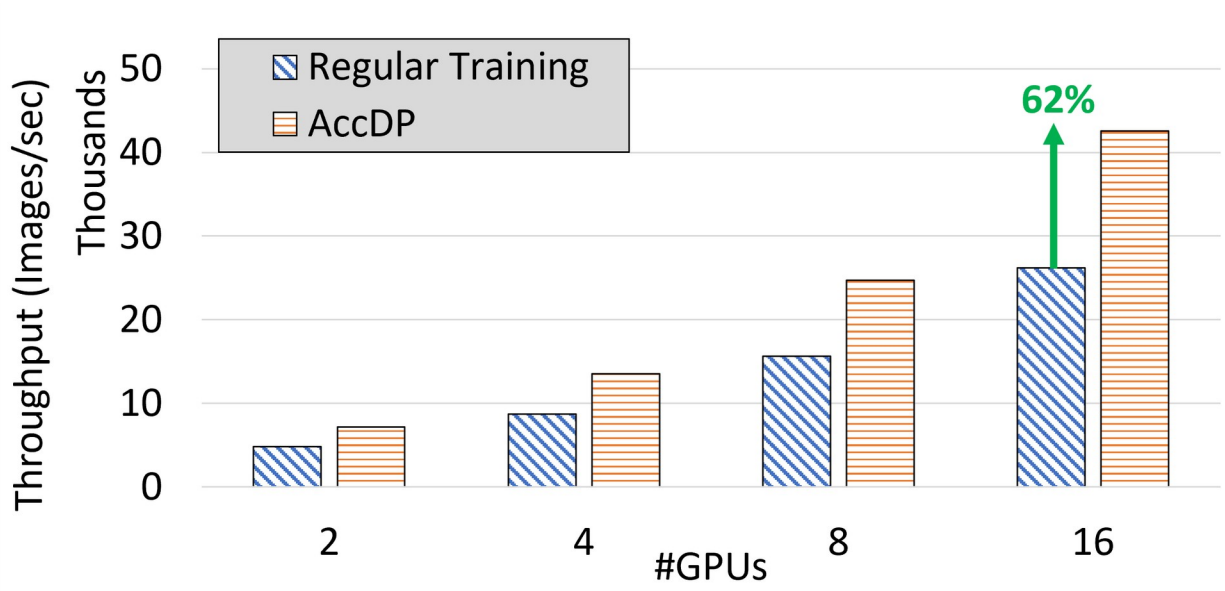
Multi node with ResNet18

- ResNet18 training throughput comparison between regular training and AccDP (proposed design) for different DNN models on up to 8 nodes 2 GPUs per node (16 GPUs) with 4 MPS clients per GPU



Multi node with ShuffleNet

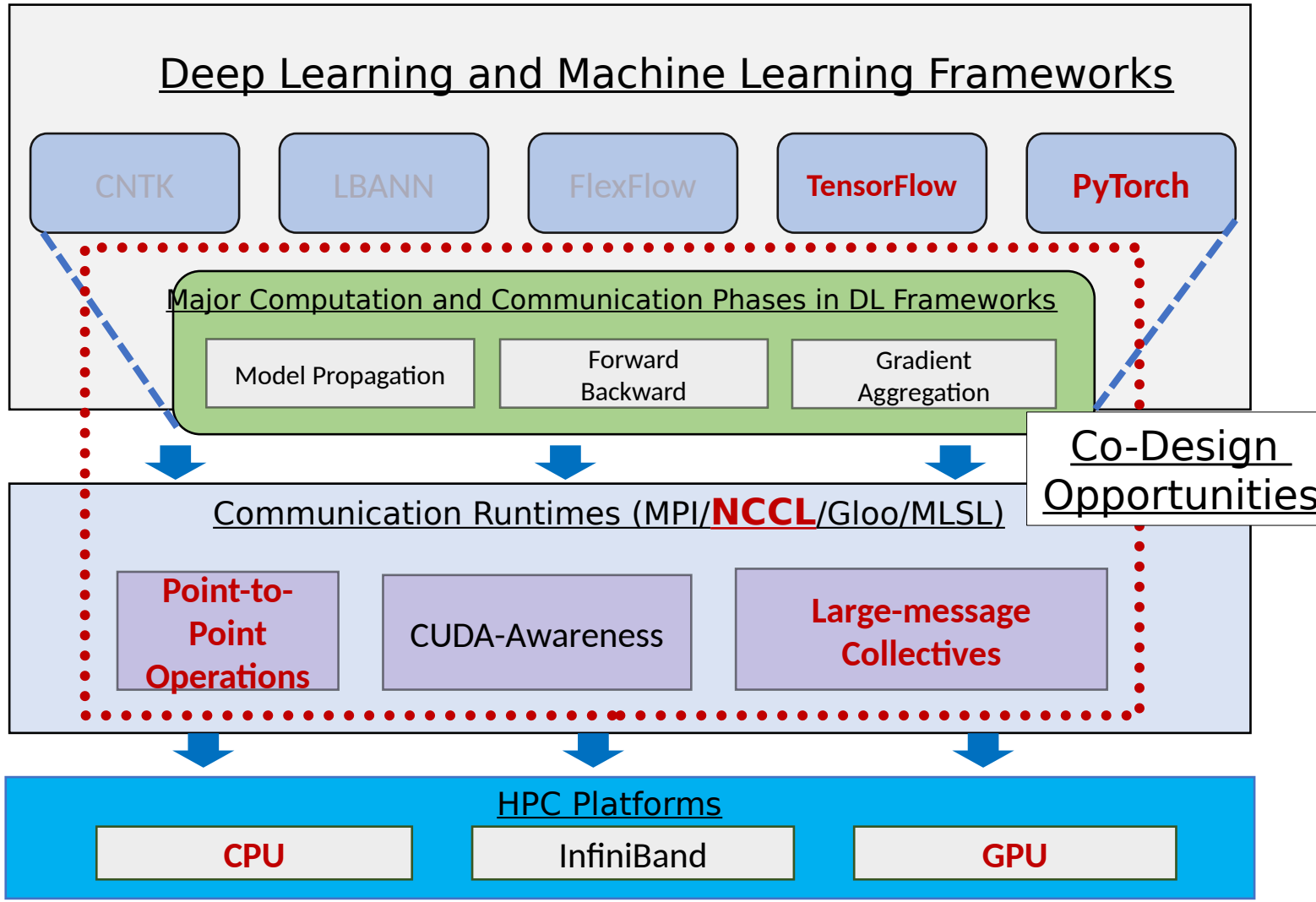
- ShuffleNet training throughput comparison between regular training and AccDP (proposed design) for different DNN models on up to 8 nodes 2 GPUs per node (16 GPUs) with 4 MPS clients per GPU.



N. Alnaasan, A. Jain, A. Shafi, H. Subramoni, and DK Panda, "AccDP: Accelerated Data-Parallel Distributed DNN Training for Modern GPU-Based HPC Clusters", HiPC'22.

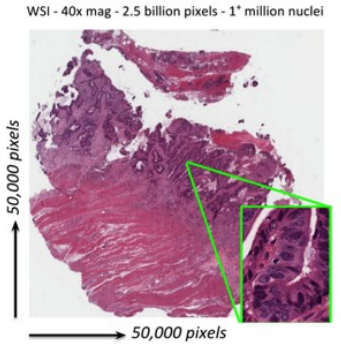
Solutions and Case Studies: Exploiting HPC for DL

- Data Parallelism
- **Model-Parallelism**

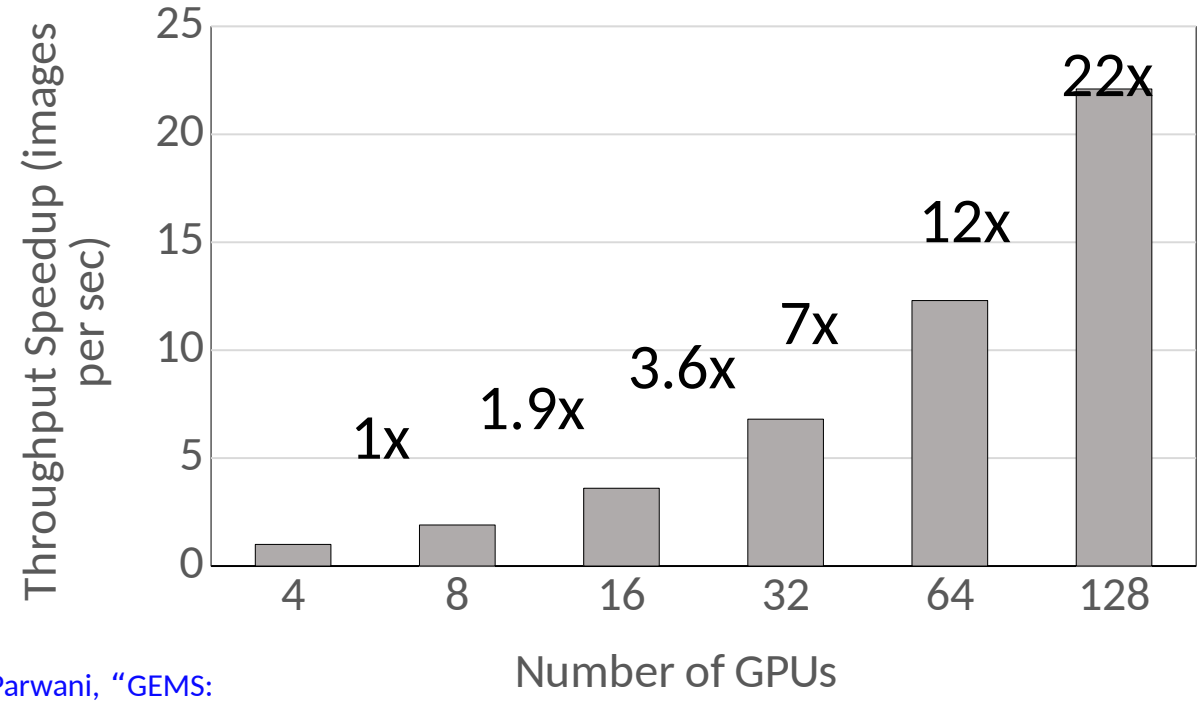


Exploiting Model Parallelism in AI-Driven Digital Pathology

- 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

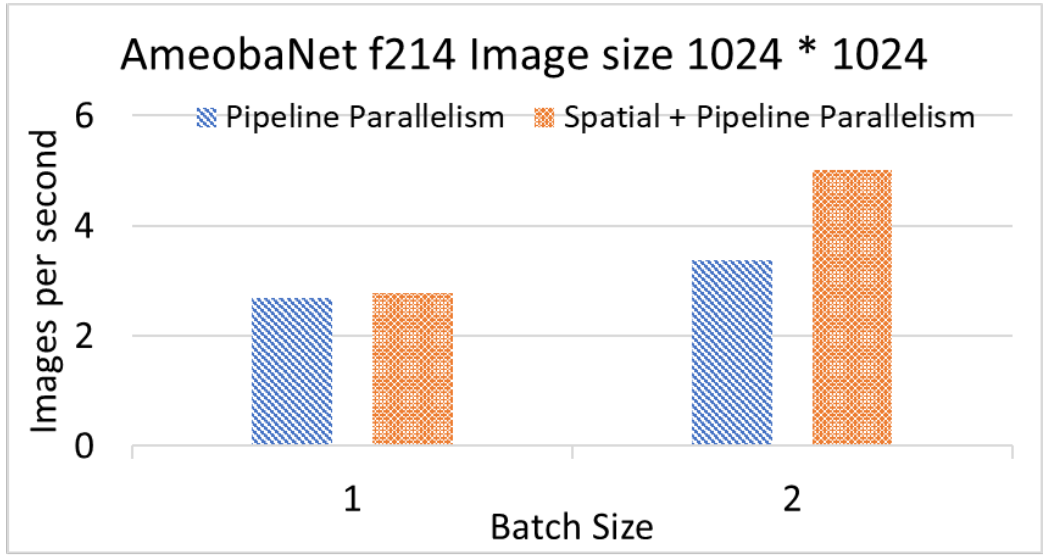
A. Jain, A. Awan, A. Aljuhani, J. Hashmi, Q. Anthony, H. Subramoni, D. K. Panda, R. Machiraju, and A. Parwani, "GEMS: GPU Enabled Memory Aware Model Parallelism System for Distributed DNN Training", Supercomputing (SC '20).

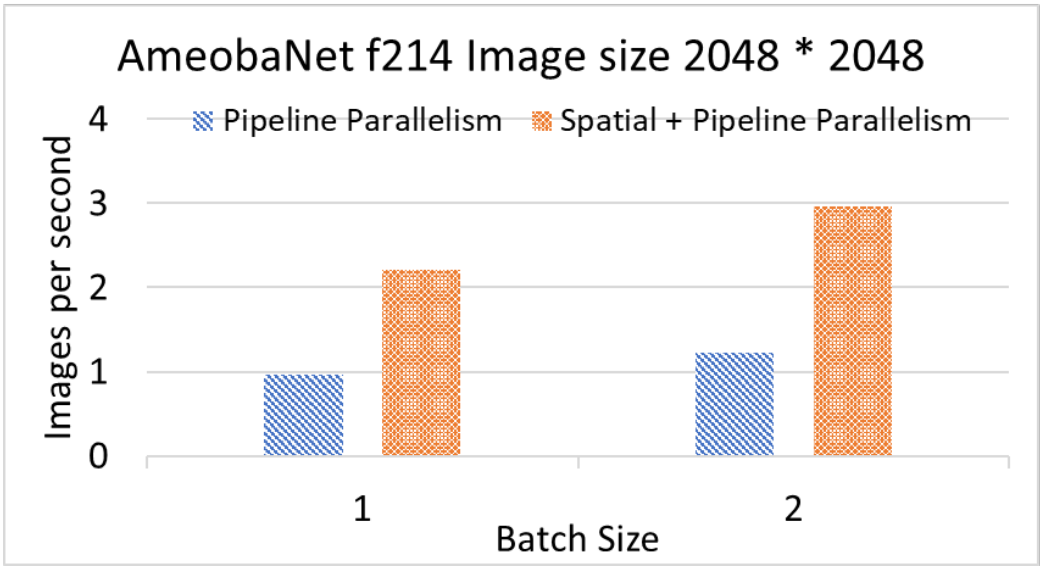
Upcoming Release: MPI4DL v0.5

MPI4DL v0.5 is a distributed and accelerated training framework for very high-resolution images that integrates Spatial Parallelism, Layer Parallelism, and Pipeline Parallelism.

Features:

- Based on PyTorch
- Support for training very high-resolution images
- Distributed training support for:
  - Layer Parallelism (LP)
  - Pipeline Parallelism (PP)
  - Spatial Parallelism (SP)
  - Spatial and Layer Parallelism (SP+LP)
  - Spatial and Pipeline Parallelism (SP+PP)
- Support for different image sizes and custom datasets.
- Exploits collective features of MVAPICH2-GDR





Outline

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

Conclusion

- Exponential growth in Machine Learning and Deep Learning frameworks
- Provided an overview of issues, challenges, and opportunities for designing efficient communication runtimes
  - Efficient, scalable, and hierarchical designs are crucial for ML and DL frameworks
  - Co-design of communication runtimes and ML and DL frameworks will be essential
- Presented use-cases to demonstrate the complex interaction between DL and ML 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.)
- 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.)
- G. Kuncham (Ph.D.)
- R. Vaidya (Ph.D.)
- J. Yao (Ph.D.)
- M. Han (M.S.)
- A. Guptha (M.S.)

Current Research Scientists

- M. Abduljabbar
- A. Shafi

Current Students (Undergrads)

- 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.)
- A. Jain (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 Senior Research Associate

- J. Hashmi

Past Programmers

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

Past Research Specialist

- M. Arnold
- J. Smith