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:**
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

Classification of Machine Learning

- Supervised Learning
- Reinforcement Learning
- Unsupervised Learning

Supervised Learning

Unsupervised Learning

Reinforcement Learning

REINFORCEMENT LEARNING

SUPERVISED LEARNING

UNSUPERVISED LEARNING

Support for Parallel and Distributed Execution

• Scikit-learn:
  − Joblib supports multi-threaded and multi-process execution (on multiple nodes)

• XGBoost:
  − Multiple ways to run on cluster of nodes:
    • Dask ([http://dask.org](http://dask.org))
    • Ray ([https://ray.io/](https://ray.io/))
    • AWS YARN
    • Apache Spark ([https://spark.apache.org/](https://spark.apache.org/)) using XGBoost4J-Spark

• cuML:
  − Execution is supposed on multiple nodes using Dask ([http://dask.org](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
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/
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)

![Diagram of a 3-layer Deep Neural Network](http://cs231n.github.io/neural-networks-1/)

Essential Concepts: Learning Rate ($\alpha$)

- **Too low**: A small learning rate requires many updates before reaching the minimum point.
- **Just right**: The optimal learning rate swiftly reaches the minimum point.
- **Too high**: Too large of a learning rate causes drastic updates which lead to divergent behaviors.

Courtesy: [https://www.jeremyjordan.me/nn-learning-rate/](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

Data Parallelism and MPI Collectives

- **Step 1: Data Propagation**
  - Distribute the data among GPUs

- **Step 2: Forward Backward Pass**
  - Perform forward pass and calculate the prediction
  - Calculate error by comparing prediction with actual output
  - Perform backward pass and calculate gradients

- **Step 3: 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

ML/DL Applications

Horovod

TensorFlow  PyTorch  MXNet

MVAPICH2 or MVAPICH2-X for CPU Training  MVAPICH2-GDR for GPU Training

PyTorch

Torch.distributed  DeepSpeed

MVAPICH2 or MVAPICH2-X for CPU Training  MVAPICH2-GDR for GPU 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-iter=5

- 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-ites=5`

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!

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.

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!!!


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

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

![Graph 1: AmeobaNet f214 Image size 1024 * 1024](chart1)

![Graph 2: AmeobaNet f214 Image size 2048 * 2048](chart2)
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 underlying HPC technologies and middleware
• Need collaborative efforts to achieve the full potential
Funding Acknowledgments

Funding Support by

[Logos of various companies]

Equipment Support by

[Logos of various companies]
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.)
- A. H. Tu (Ph.D.)
- G. Kuncham (Ph.D.)
- P. Kousha (Ph.D.)
- S. Xu (Ph.D.)
- R. Vaidya (Ph.D.)
- B. Michalowicz (Ph.D.)
- Q. Zhou (Ph.D.)
- J. Yao (Ph.D.)
- B. Ramesh (Ph.D.)
- K. Al Attar (M.S.)
- M. Han (M.S.)
- K. K. Suresh (Ph.D.)
- L. Xu (Ph.D.)
- A. Gupta (M.S.)

**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. Nororina (Ph.D.)
- X. Ouyang (Ph.D.)
- S. Pai (M.S.)
- S. Potluri (Ph.D.)
- K. Raj (M.S.)
- R. Rajachandrasekar (Ph.D.)

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

**Current Research Scientists**
- M. Abduljabbar
- A. Shafi

**Current Research Specialist**
- R. Motlagh

**Current Research Scientists**
- M. Abduljabbar
- A. Shafi

**Current Students (Undergrads)**
- T. Chen

**Current Software Engineers**
- B. Seeds
- N. Pavuk
- N. Shineman
- M. Lieber

**Current Faculty**
- H. Subramoni

**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

**Current Research Specialist**
- R. Motlagh

**Current Faculty**
- H. Subramoni

**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