Distributed Deep Learning on Summit

October 2019 OLCF User Conference Call

Bryant Nelson, Brad Nemanich

October 30, 2019

Deep Learning on Summit

- Deep learning training jobs run particularly well on Summit due to the large number of fast GPUs and the high speed interconnection between them
- However, there are some road blocks to getting started
  - There are many different deep learning frameworks, but they may need to be built in a particular way to run well on Summit
  - A typical deep learning job relies on many different packages and components. It can be difficult to install all of the dependencies and verify they can all work within the same environment
  - Frameworks are very new, and constantly changing, so troubleshooting can be difficult
  - It can be difficult to run deep learning jobs that are distributed across multiple nodes.

Watson Machine Learning CE

- Provides a curated set of pre-built deep learning packages that have been optimized to run on IBM Power systems with Nvidia GPUs
- All packages within WML CE have been verified to work well within the same environment
- These packages are fully supported on Summit by IBM
- Includes IBM Distributed Deep Learning (DDL), which provides easy to use tools to run deep learning jobs across a cluster and has been tuned to run well on Summit on up to 954 nodes
- Provides other unique features from IBM such as Large Model Support (LMS) and SnapML

Watson Machine Learning CE

Curated, tested and pre-compiled binary software distribution that enables enterprises to quickly and easily deploy deep learning for their data science and analytics development

Including all of the following frameworks:

Snap ML

Bazel

HDF5

ONNX

OpenBLAS

Caffe Caffe2

TensorFlow
TensorFlow Probability
TensorBoard
TensorFlow-Keras

BVLC Caffe
IBM Enhanced Caffe
Caffe2

PyTorch

Watson Machine Learning CE 1.6.1

caffe	IBM-optimized version of Berkeley Vision and Learning Center Caffe	1.0.0
caffe-cpu	IBM-optimized Caffe CPU-only package	1.0.0
cudf	Rapids cuDF	0.7.2
cuml	Rapids cuML	0.7.0
ddl	Distributed Deep Learning	1.4.0
horovod	Horovod built with IBM DDL (Summit Only)	0.16.4
pai4sk	WML CE Snap ML	1.4.0
pytorch	PyTorch	1.1.0
snapml-spark	WML CE Snap ML Spark	1.3.0
tensorflow	TensorFlow CPU-only package	1.14
tensorflow-gpu	TensorFlow with GPU support	1.14
tensorflow-serving	TensorFlow Serving CPU-only package	1.14
tensorflow-serving-gpu	TensorFlow Serving with GPU support	1.14
tensorflow2-gpu	TensorFlow with GPU support	2.0 (beta)
xgboost	xgboost with GPU support	18.04
xgboost-cpu	xgboost CPU-only package	18.04

Watson Machine Learning CE: Large Model Support

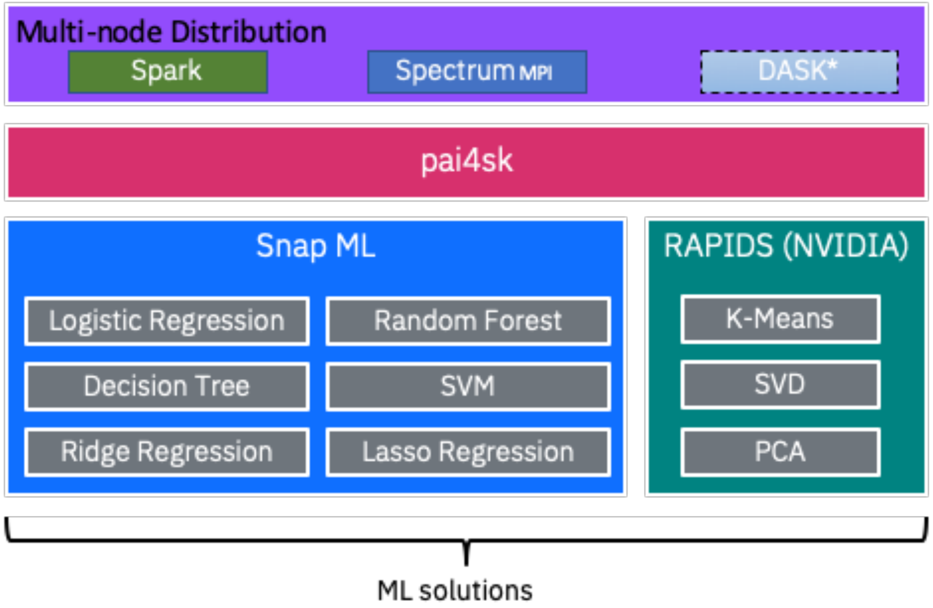
- Seamlessly moves layers of a model between the GPU and CPU to overcome GPU memory limits
- Allows training of:
  - Deeper models
  - Higher resolution data
  - Larger batch sizes
- Summit nodes have a fast NVLink 2.0 connection between the CPU and GPU, which allows for data swapping with minimal overhead

Watson Machine Learning CE: Large Model Support

- Large Model Support is enabled for Watson Machine Learning CE's versions of TensorFlow, PyTorch and Caffe
- For more information, see: https://developer.ibm.com/linuxonpower/2019/06/11/tensorflow-large-model-support-resources/

Watson Machine Learning CE: SnapML

- Parallel and accelerated machine learning algorithms
- SnapML is packaged as an extension to scikit-learn



* Future support

Watson Machine Learning CE: Conda Channel

- WML CE is provided as a collection of conda packages
- A module exists on Summit which contains WML CE
- WML CE is also provided as a public conda channel at: https://public.dhe.ibm.com/ibmdl/export/pub/software/server/ibm-ai/conda/
- An x86 version of WML CE also exists on the public conda channel. This allows for users to begin development on a workstation, then easily move to Summit.

Watson Machine Learning CE: Summit Module

- The WML CE Module can be loaded on Summit with the command
  - module load ibm-wml-ce
- Loading the module will:
  - Activate a conda environment that has all of Watson Machine Learning CE installed
  - Setup conda channels that point to IBM's public conda channel for WML CE, as well as a local conda channel with Summit specific packages
- Users can either use the activated conda environment, or create their own environments from the channels provided by the module

Watson Machine Learning CE: Running

- Running a simple python script that uses Watson Machine Learning CE

```
$ bsub -P PROJ -W 0:20 -nnodes 1 -Is $SHELL
Job <701106> is submitted to default queue <batch>.
<<Waiting for dispatch ...>>
<<Starting on batch1>>
bash-4.2$ module load ibm-wml-ce
(ibm-wml-ce-1.6.1) bash-4.2$ python my_deep_learning_script.py
```

Watson Machine Learning CE: Installing Other Packages

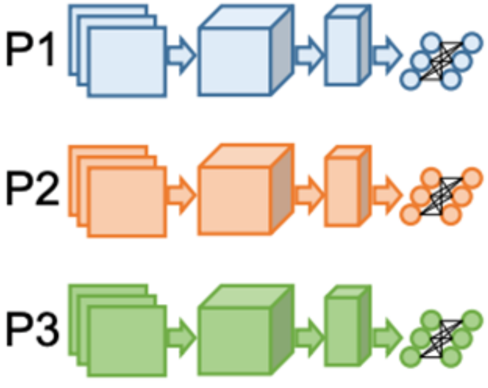
- The Watson Machine Learning CE default conda environment is read-only. If additional packages are required, the module's environment can be cloned with the following commands:

```
$ module load ibm-wml-ce
(ibm-wml-ce-1.6.1) $ conda create --name cloned_env --clone ibm-wml-ce-1.6.1
...
(ibm-wml-ce-1.6.1) $ conda activate cloned_env
(cloned_env) $
```

- Note: By default, this will install the new conda environment within the user's home directory
- For more information please see: https://docs.olcf.ornl.gov/software/analytics/ibm-wml-ce.html

Data Parallel Distributed Deep Learning: Description

- Synchronous All-to-All Data-Parallel Distributed GPU Deep Learning
- A process is created for each GPU in the cluster
- Each process contains a complete copy of the model
- Mini-batch is spread across all of the processes
  - Each process uses different input data
- After each iteration, all of the processes sync and average together their gradients, and those averages are used to update the local weights.
- Models on each GPU should always be identical.



(a) Data Parallelism

[Bergstra et al. 2012]

Data Parallel Distributed Deep Learning: Tools

- Communication Libraries
  - MPI
  - NCCL
  - IBM DDL
- Integrations / Frameworks
  - TensorFlow Distribution Strategies
  - Horovod
  - IBM DDL

Data Parallel Distributed Deep Learning: Tools – Communication Libraries

- The following tools are libraries, which provide the communication functions necessary to perform distributed training. Primarily allReduce and broadcast functions.
  - MPI
    - Classic tool for distributed computing.
    - Still commonly used for distributed deep learning.
  - NCCL
    - Nvidia's gpu-to-gpu communication library.
    - Since NCCL2, between-node communication is supported.
  - IBM DDL
    - Provides a topology-aware allReduce.
    - Capable of optimally dividing communication across hierarchies of fabrics.
    - Utilizes different communication protocols at different hierarchies.

Data Parallel Distributed Deep Learning: Tools – Integrations / Frameworks

- The following tools are libraries, which provide integrations into deep learning frameworks to enable distributed training using common communication libraries.
  - TensorFlow Distribution Strategies
    - Native Tensorflow distribution methods.
  - Horovod [Sergeev et al. 2018]
    - Provides integration libraries into common frameworks which enable distributed training with common communication libraries, including
  - IBM DDL
    - Provides integrations into common frameworks, including a Tensorflow operator that integrates IBM DDL with Tensorflow.

Data Parallel Distributed Deep Learning: IBM DDL [Cho et al. 2016]

- IBM DDL provides:
  - C and Python libraries that provide communication functions.
    - The library utilizes the MPI and NCCL libraries
  - Framework integrations
    - Provides a custom operator for TensorFlow. To use in Tensorflow, only need to ‘import ddl’.
    - DDL integration is built into WML-CE’s version of Caffe and PyTorch
    - WML CE provides a version of Horovod that is built with DDL
  - A tool for launching jobs across a cluster called ddlrun, simplifying the launching of distributed jobs.
    - e.g. ddlrun -H server1,server2,server3,server4 python train.py
- DDL’s allReduce uses knowledge of the cluster layout to perform reductions between nodes in a certain order
  - DDL attempts to perform reductions between nodes in the order that will cause the lowest communication overhead.
  - It takes into account the fact that not all nodes are connected with the same interface
  - DDL performs best compared to other allreduce libraries when used in a cluster with a non-flat topology.

Data Parallel Distributed Deep Learning: IBM DDL Example

Steps to distribute the training of a tf.keras model:

1. Import the ddl library.
2. Split the training data.
3. Modify hyperparameters.
4. Add callbacks.

We will go through the changes necessary to use DDL to distribute the training of an mnist model in tf.keras.

Original script: https://github.com/keras-team/keras/blob/4f2e65c385d60fa87bb143c6c506cbe428895f44/examples/mnist_cnn.py

Data Parallel Distributed Deep Learning: IBM DDL Example

1. Import the ddl library.

This is the only ***necessary*** step to enable distributed training with DDL.

```
| from tensorflow.python import keras as keras
| from tensorflow.python.keras.datasets import mnist
| from tensorflow.python.keras.models import Sequential
| from tensorflow.python.keras.layers import Dense, Dropout, Flatten
| from tensorflow.python.keras.layers import Conv2D, MaxPooling2D
| from tensorflow.python.keras import backend as K
> import ddl
> import numpy as np
```

Data Parallel Distributed Deep Learning: IBM DDL Example

2. Split the training data.

- If training works by iterating over all of the data, each process should only iterate over equal sections of the data.

- If training works by grabbing random data, modifications may not be necessary, although it should be verified that a different seed is being used for each process

Data Parallel Distributed Deep Learning: IBM DDL Example

2. Split the training data.

```
> # DDL: Save the full test data before splitting for final accuracy check.
> x_test_full = x_test.astype('float32') / 255
> y_test_full = keras.utils.to_categorical(y_test, num_classes)
>
> # DDL: Split the training & testing data.
> x_train = np.array_split(x_train, ddl.size())[ddl.rank()]
> x_test = np.array_split(x_test, ddl.size())[ddl.rank()]
  x_train = x_train.astype('float32')
  x_test = x_test.astype('float32')
  x_train /= 255
  x_test /= 255
  print('x_train shape:', x_train.shape)
  print(x_train.shape[0], 'train samples')
  print(x_test.shape[0], 'test samples')

> # DDL: Split the training & testing data.
> y_train = np.array_split(y_train, ddl.size())[ddl.rank()]
> y_test = np.array_split(y_test, ddl.size())[ddl.rank()]
```

Data Parallel Distributed Deep Learning: IBM DDL Example

3. Modify hyperparameters.

- In this example the only hyperparameter we change is the learning rate.
  - We scale the learning rate by the total number of "learners" to offset the effect of the larger global batch size.

```
> # DDL: adjust learning rate based on number of GPUs.
  model.compile(loss=keras.losses.categorical_crossentropy,
|                   optimizer=keras.optimizers.Adadelta(lr=1.0 * ddl.size()),
                    metrics=['accuracy'])
```

Data Parallel Distributed Deep Learning: IBM DDL Example

4. Add callbacks.

- DDL provides two tf.keras callbacks.
  - ddl.DDLCallback() is responsible for synchronizing keras metrics
    - Should always be the first callback in the callbacks list.
  - ddl.DDLGlobalVariablesCallback() is responsible for initializing global variables to the same values across all learners
    - Should always be the last callback in the callbacks list.

```
> callbacks = list()
>
> # DDL: Add the DDL callback.
> callbacks.append(ddl.DDLCallback())
> callbacks.append(ddl.DDLGlobalVariablesCallback())
```

Data Parallel Distributed Deep Learning: IBM DDL Example

- Execution
- DDL provides a utility called DDLRUN which is used to launch the learning job on any number of nodes/gpus.

```
(demo) [bnelson@dlw12 ~]$ ddlrun -H dlw03  python \ ~/anaconda3/envs/demo/tf_cnn_benchmarks/tf_cnn_benchmarks.py --variable_update=ddl \
--model=resnet50 --num_gpus=1 --batch_size=32
...
----------------------------------------------------------------
total images/sec: 1248.06
----------------------------------------------------------------


(demo) [bnelson@dlw12 ~]$ ddlrun -H dlw04,dlw05,dlw06,dlw07,dlw08,dlw09,dlw10,dlw11,dlw12,dlw13 \ python ~/anaconda3/envs/demo/tf_cnn_benchmarks/tf_cnn_benchmarks.py --variable_update=ddl \
--model=resnet50 --num_gpus=1 --batch_size=32
...
----------------------------------------------------------------
total images/sec: 12043.78
----------------------------------------------------------------
```

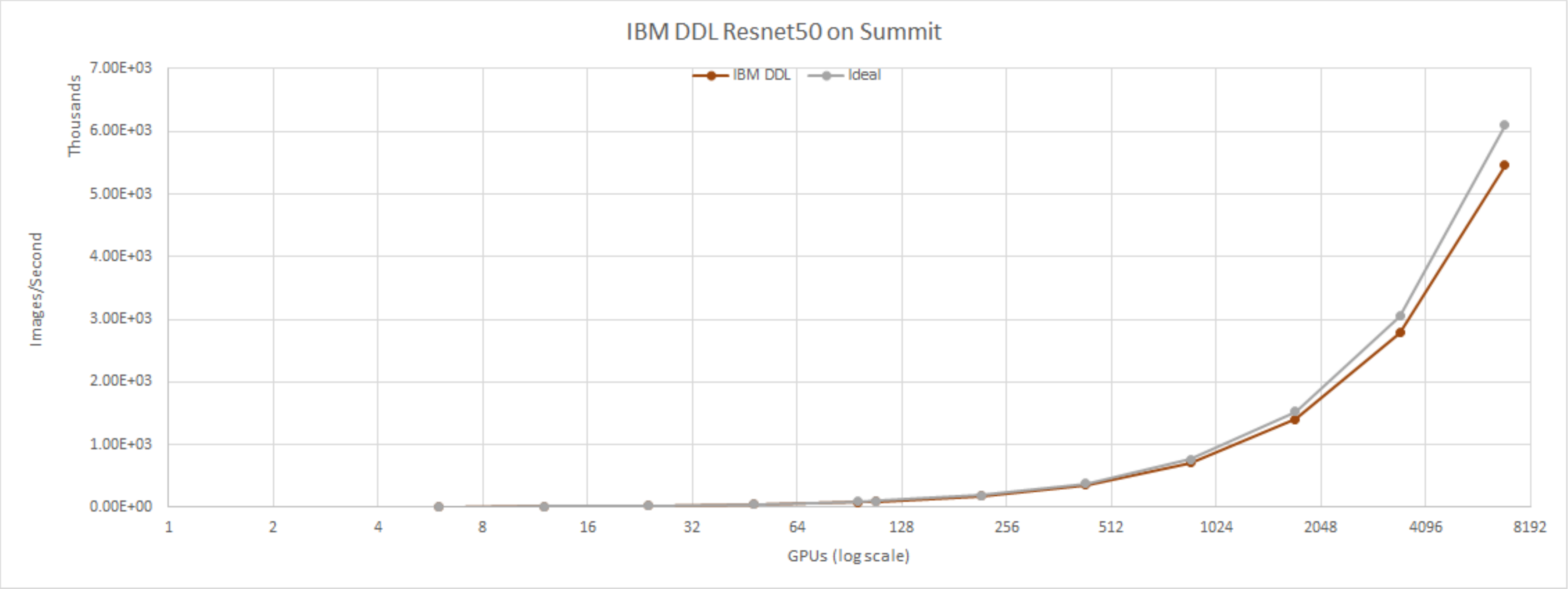
Data Parallel Distributed Deep Learning: IBM DDL Example - Summit BSUB Script

- This BSUB script will launch an 18 node DDL training job on Summit:

```
#!/bin/bash
#BSUB -W 0:20
#BSUB -P <project>
#BSUB -q batch
#BSUB -J ddl_test
#BSUB -nnodes 18
module load ibm-wml-ce

ddlrun python $CONDA_PREFIX/tf_cnn_benchmarks/tf_cnn_benchmarks.py \
--variable_update=horovod \
--model=resnet50 \
--num_gpus=1 \
--batch_size=256 \
--num_batches=100 \
--num_warmup_batches=10 \
--data_name=imagenet \
--allow_growth=True \
--use_fp16
```

IBM DDL Example - Performance On Summit



Watson Machine Learning CE: Horovod

- The ibm-wml-ce module on Summit also provides Horovod built with the IBM DDL backend.
- This version of Horovod should work with existing deep learning scripts built to use Horovod.
- For more information see: https://github.com/horovod/horovod

Data Parallel Distributed Deep Learning: Technical Considerations

- Batch Size
- Learning Rate
- Batch Normalization
- On-The-Fly Validation
- Data Pipelining

For More Information

- Documentation is available at:
  https://docs.olcf.ornl.gov/software/analytics/ibm-wml-ce.html
- For support either:
  – Call 865-241-6536
  – Email: help@olcf.ornl.gov

Thank You

Bryant Nelson
Brad Nemanich

—

bryant.nelson@ibm.com
brad.nemanich@ibm.com
http://www.ibm.com