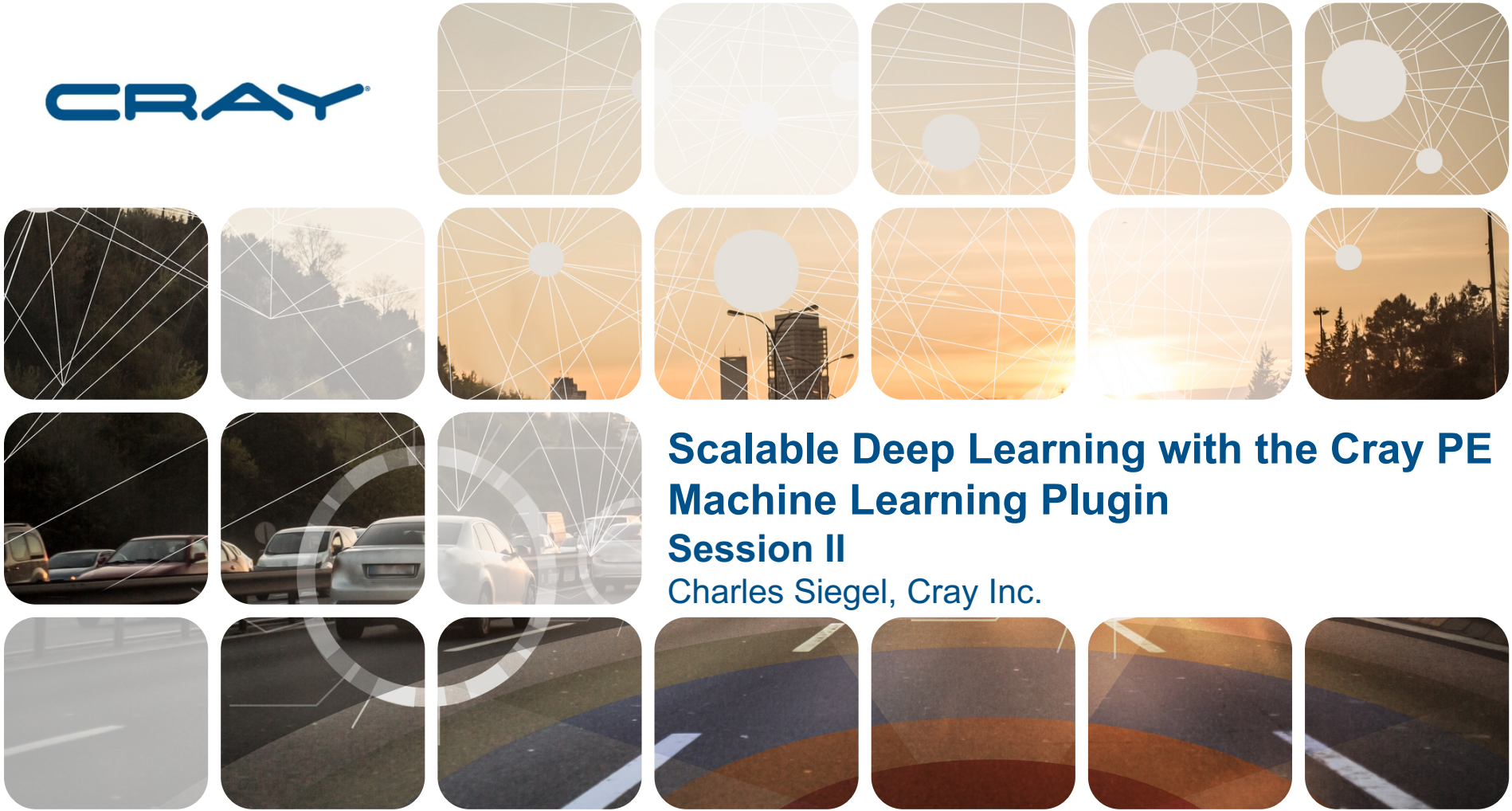
Analytics and AI on Cray Systems

Session II



Scalable Deep Learning with the Cray PE Machine Learning Plugin

Session II

Charles Siegel, Cray Inc.

HPC Attributes of Deep Learning

HPC Attributes

- **DL training is a classic high-performance computing problem which demands:**
  - Large compute capacity in terms of FLOPs, memory capacity and bandwidth
  - A performant interconnect for fast communication of gradients and model parameters
  - Parallel I/O and storage with sufficient bandwidth to keep the compute fed at scale

Data Parallelism - Collective-based Synchronous SGD

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

Algorithm 1 Sync-SGD algorithm

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

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

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

APPLY_GRADIENTS(G_{global})

end for

Compute intensive

Communication intensive

Typically not much compute

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

Why do we want to scale?

- **Deep Network Training**
  - We can strong scale training time-to-accuracy provided
    - Number of workers (e.g., # nodes) << number of training examples
    - Learning rate for particular batch size / scale is known
- **Hyper-Parameter Optimization**
  - For problems and datasets where baseline accuracy is not known
    - learning rate schedule
    - momentum
    - batch size
  - Evolve topologies if good architecture is unknown (common with novel datasets / mappings)
    - Layer types, width, number filters
    - Activation functions, drop-out rates

Parallelization Methods for DL

Parallelization Techniques

- **Data Parallelism**
  - As described earlier, divides global mini-batch among processes
  - Two methods for this:
    - Synchronous: single model (possibly replicated across all processes) updated with globally averaged gradients every iteration
    - Asynchronous: processes provide gradients every iteration but are allowed to fall out of sync from one another. Processes each have their own model that may or may not be the same as any other process
- **Model Parallelism**
  - Single model with layers decomposed across processes
  - Activations communicated between processes
- **Examples will focus on synchronous data parallel approach**

Distributed TensorFlow

- **TensorFlow has a native method for parallelism across nodes**
  - ClusterSpec API
  - Uses gRPC layer in TensorFlow based on sockets
- **Can be difficult to use and optimize**
- **User must specify**
  - hostnames and ports for all worker processes
  - hostnames and ports for all parameter server processes (see next slide)
  - # of workers
  - # of parameter server processes
  - Chief process of workers

Distributed TensorFlow

- **Number of parameter servers (PS) processes to use is not clear**
  - Too few PS results in many-to-few communication pattern (very bad) and stalls delivering updated parameters
  - Too many PS results in many-to-many communication pattern (also bad)
- **Users typically have to pick a scale and experiment for best performance**

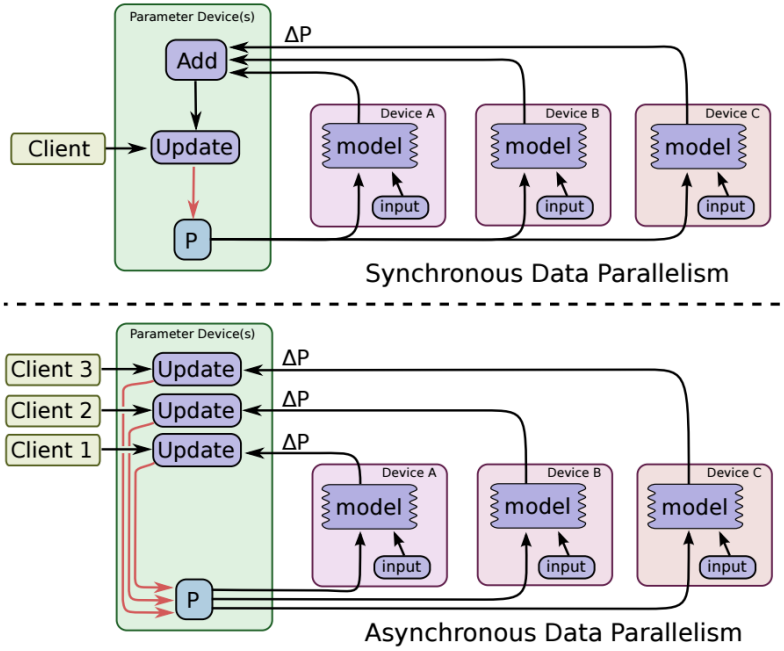
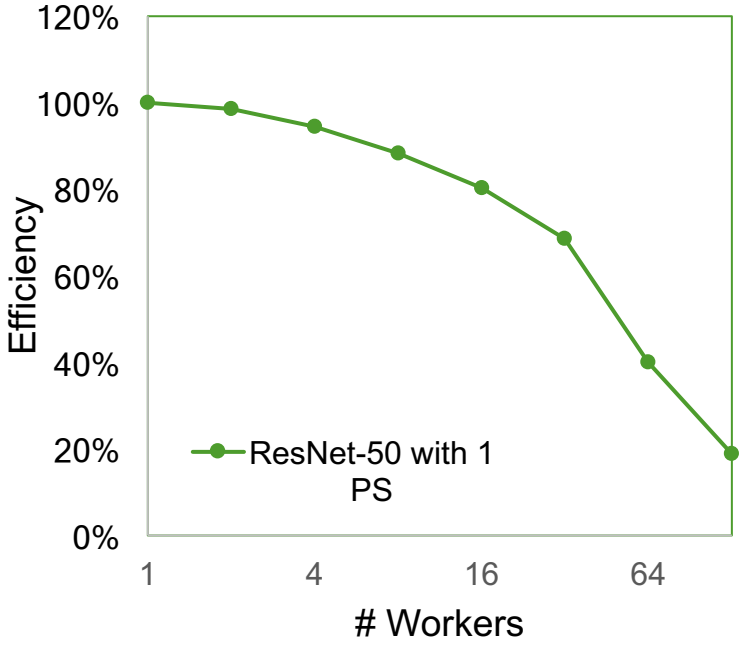
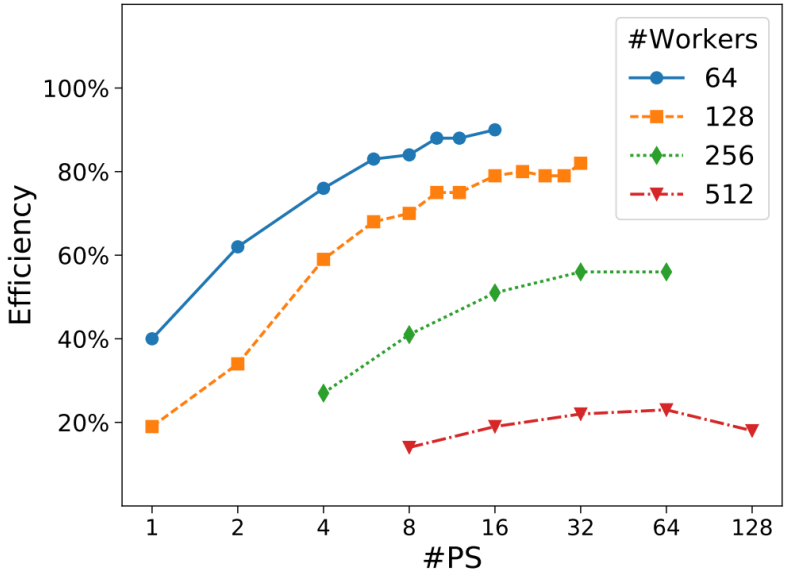


Figure 7: Synchronous and asynchronous data parallel training

Distributed TensorFlow Scaling on Cray XC40 - KNL



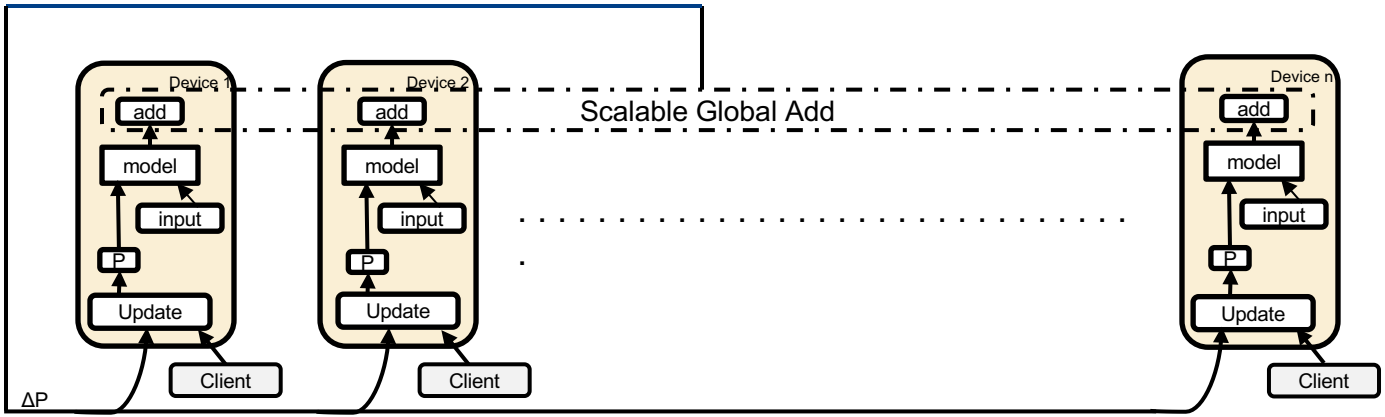


From Mathuriya et al. @ NIPS 2017

MPI-based Data Parallel TensorFlow

- The performance and usability issues with distributed TensorFlow can be addressed by adopting an MPI communication model
- TensorFlow does have an MPI option, but it only replaces point to point operations in gRPC with MPI
  - Collective algorithm optimization in MPI not used
- Other frameworks, such as Caffe and CNTK, include MPI collectives
- An MPI collective based approach would eliminate the need for PS processes and likely be optimized without intervention from the user

Scalable Synchronous Data Parallelism



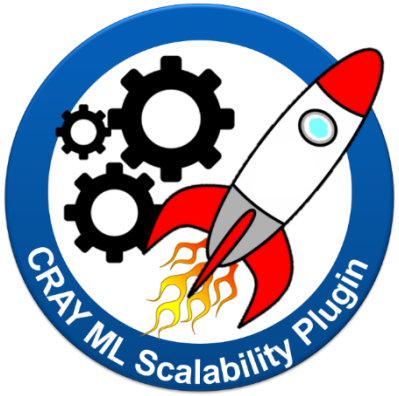
- Note there are no PS processes in this model
- Resources dedicated to gradient calculation

Uber Horovod

- **Uber open source addon for TensorFlow *only* that replaces native optimizer class with a new class**
  - Horovod adds an allreduce between gradient computation and model update in this class
- **New Python class includes NCCL and MPI collective reductions for gradient aggregation**
- **https://github.com/uber/horovod**
- **No modifications to TensorFlow source required**
  - User modifies Python training script instead

Cray Programming Environment Machine Learning Plugin (CPE ML Plugin)

- **DL communication plugin with Python and C APIs**
- **Optimized for TensorFlow but also portable to other frameworks**
  - Callable from C/C++ source
  - Called from Python if data stored in NumPy arrays or Tensors
- **Like Horovod does not require modification to TensorFlow source**
  - User modifies training script
- **Uses custom allreduce specifically optimized for DL workloads**
  - Optimized for Cray Aries interconnect and IB for Cray clusters
- **Tunable through API and environment variables**
- **Supports multiple gradient aggregations at once with thread teams**
  - Useful for Generative Adversarial Networks (GAN), for example



CPE ML Plugin Example

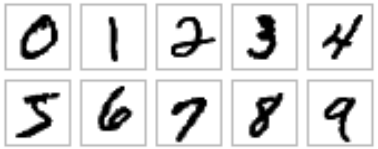
Tensorflow Training Script Modifications

- **Both Horovod and CPE ML Plugin require some modifications to a serial training script**

- **For the CPE ML Plugin the changes are**
  - Importing the Python module (ml_comm)
  - Initialize the module
    - Possibly configure the thread team(s) for specific uses
  - Broadcast initial model parameters
  - Possible modifications to learning rate decay schedules and other mini-batch size dependent parameters to account for the effective mini-batch size across all processes
  - Incorporate gradient aggregation between gradient computation and model update
  - Finalize the Python module

MNIST Example

- Dataset of handwritten digits from 0-9
- Simple CNN can be used to identify handwritten digits



- This example is adapted from the TensorFlow official MNIST example
- https://github.com/tensorflow/models/tree/master/official/mnist
- Modified script included with CPE ML Plugin
  - `module load craype-ml-plugin-py2/1.1.0`
  - `less $CRAYPE_ML_PLUGIN_BASEDIR/examples/tf_mnist/mnist.py`

CPE ML Plugin - Import

- Access the Python API by importing the module

```
import tensorflow as tf

# CRAY ADDED
import ml_comm as mc
import math
#
```

CPE ML Plugin - Initialization

- **Compute the number of trainable variables in the model**
  - Required for the CPE ML Plugin to pre-allocate needed communication buffers
- **Example for init sets up a single thread team with one local thread per team, per MPI rank**
  - For GANs, may want to use 2 teams, to have 2 reductions in flight
  - Number of threads per team, tunable parameter, biggest gain (1 to 2)

```
# CRAY ADDED
if FLAGS.enable_ml_comm:

  # initialize the Cray PE ML Plugin (assume 20M variables max)
  totsize = sum([reduce(lambda x, y: x*y, v.get_shape().as_list()) for v in tf.trainable_variables()])
  mc.init(1, 1, totsize, "tensorflow")
```

CPE ML Plugin – Team Configuration

- **Set the maximum number of steps (mini batches) to train for**
  - Verbose output every 200 steps
- **Also set output path to rank-specific location**

```
# config the thread team (correcting the number of epochs for the effectice batch size))
FLAGS.train_epochs = int(FLAGS.train_epochs / mc.get_nranks())
max_steps = int(math.ceil(FLAGS.train_epochs *
                (_NUM_IMAGES['train'] + _NUM_IMAGES['validation']) / FLAGS.batch_size))
mc.config_team(0, 0, 100, max_steps, 2, 200)

# give each rank its own directory to save in
FLAGS.model_dir = FLAGS.model_dir + '/rank' + str(mc.get_rank())
```

CPE ML Plugin – Broadcast Initial Model

- Broadcast initial model parameter values from rank 0 to all other ranks
- Then assign broadcasted values locally

```
# CRAY ADDED
# since this script uses a monitored session, we need to create a hook to initialize
# variables after the session is generated
class BcastTensors(tf.train.SessionRunHook):

  def __init__(self):
    self.bcast = None

  def begin(self):
    if not self.bcast:
      new_vars   = mc.broadcast(tf.trainable_variables(),0)
      self.bcast = tf.group(*[tf.assign(v,new_vars[k]) for k,v in enumerate(tf.trainable_variables())])
```

CPE ML Plugin – Gradient Aggregation

- Perform gradient averaging across all ranks between local gradient calculation and model update

```
# CRAY ADDED
if FLAGS.enable_ml_comm:

  # we need to split out the minimize call below so we can modify gradients
  grads_and_vars = optimizer.compute_gradients(loss)

  grads      = mc.gradients([gv[0] for gv in grads_and_vars], 0)
  gs_and_vs = [(g,v) for (_,v), g in zip(grads_and_vars, grads)]

  train_op = optimizer.apply_gradients(gs_and_vs,
                                       global_step=tf.train.get_or_create_global_step())
# END CRAY ADDED
```

CPE ML Plugin – Finalize

- After all training steps are complete clean up data structures and MPI

```
# CRAY ADDED
if FLAGS.enable_ml_comm:
  mc.finalize()
# END CRAY ADDED
```

CPE ML Plugin – More information

- **module avail craype-ml-plugin**
  - craype-ml-plugin-py2/1.0.1(TF 1.3)
  - craype-ml-plugin-py3/1.0.1(TF 1.3)
- **Load modules in order:**
  - module load cray-python
  - module load craype-ml-plugin-py3

- **Please refer to CPE ML Plugin manpage for more details on usage**
  - `man intro_ml_plugin`
  - `Or from python shell`

```
$ python3
>>> import ml_comm as mc
>>> help(mc)
>>> help(mc.init)
```

- **Examples directory**
  - $CRAYPE_ML_PLUGIN_BASEDIR/examples

CPE ML Plugin – Execution using Urika-XC

Procedure:

1. Load the **analytics** module

```
$ module load analytics
```

2. Allocate interactive nodes via SLURM or PBS

```
$ salloc --nodes=2 --exclusive --gres=gpu -C P100
```

```
$ qsub –I –q p100 –lnodes=2
```

3. I

```
$ run_training –n 2 --ppn 1 --cudnn-libs /path/to/cudnn-8.0-v51/cuda/lib64
  --no-node-list “python mnist.py --enable_ml_comm
  --data_dir=[mnist data pth] --model_dir=[train dir]”
```

CPE ML Plugin Execution – Native XC

Procedure:

1. Locate/Install Tensorflow and set PYTHONPATH
2. Load the module cray-python and the craype-ml-plugin modules

```
$ module load cray-python
$ module load craype-ml-plugin.py3
```

3. Allocate interactive nodes (examples uses SLURM)

```
$ salloc --nodes=2 --exclusive --gres=gpu -C P100
```

4. Execute the modified Tensorflow training script using srun

```
$ srun --ntasks=2 --ntasks-per-node=1 --cpu_bind=none  python3 mnist.py
   --enable_ml_comm --data_dir=[mnist data pth] --model_dir=[train dir]
```

Instructions for Running Tensorflow on Piz Daint

% module load daint-gpu
% module avail Tensorflow
TensorFlow/1.2.1-CrayGNU-17.08-cuda-8.0-python3(default)
TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
TensorFlow/1.4.1-CrayGNU-17.08-cuda-8.0-python3
TensorFlow/1.4.1-CrayGNU-17.12-cuda-8.0-python3
TensorFlow/1.7.0-CrayGNU-17.12-cuda-8.0-python3
% module load TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
Loads cray-python, cuDNN, etc
% module load craype-ml-plugin-py3/1.0.1

```
$ salloc --nodes=2 --exclusive --constraint=gpu
```

Note: SLURM option --constraint=gpu on Piz Daint allocates the XC50 Intel Haswell 12-core nodes with GPU devices and automatically sets the SLURM option **--gres=gpu**

Class Exercise: Tensorflow + CPE ML Plugin

- **Run MNIST model using CPE ML Plugin on Piz Daint**
- **Step-by-Step Instructions:**
  - /scratch/snx3000/kristyn/CUG2018/Tensorflow/README
  - Script provided to convert raw MNIST data to TFRecords file format, but can also use
    - --data_dir=/scratch/snx3000/kristyn/datasets/mnist_data

Using CPE ML Plugin with Keras

- **Instructions for installing Keras**
  - Note: need to make sure Tensorflow and Keras versions are compatible
  - For Tensorflow 1.3.0 need to use Keras 2.1.5
    - git clone https://github.com/keras-team/keras.git
    - cd keras
    - git checkout tags/2.1.5
  - Note:
    - Also make sure to use the Keras examples from the 2.1.5 branch

Using CPE ML Plugin with Keras

- **Modifications:**
  - Leverage the existing Tensorflow backend to keras
    - Modify backend file to include CPE ML modifications
      - keras-2.1.5/keras/keras/backend/tensorflow_backend.py
    - Recompile
  - Small modifications to user-level scripts
    - Passing information to the backend for initialization, team configuration, and finalize

- **Thanks to Diana Moise!**

Instructions for installing Keras (cont)

- **Native XC**

  % module load cray-python
  % module load TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
  % python3 setup.py install --user

- **Inside Urika-XC container**
  - Install into root: cd to keras directory, run install

    % python setup.py install --user

  - Install into conda environment

    % conda create --clone py36_tf_cpu --name py36_keras_cray_ml
    % python setup.py install –user
    % export PYTHONPATH=/opt/tensorflow_cpu:$PYTHONPATH

Keras + CPE ML Plugin: Modifications to backend

- **Added to tensorflow_backend.py**
  - Import ml_comm library and add new functions

```
# CRAY ADDED
import ml_comm as mc

# threads per team, number of teams, algorithm to use, grads len, total steps, ksteps, v, freq
def mc_init(th_team, n_teams, alg, grads_len, total_steps, ksteps, v, freq):
    mc.init(th_team, n_teams, grads_len, "tensorflow")
    for team in range(n_teams):
        mc.config_team(team, alg, ksteps, total_steps, v, freq)

def mc_get_rank():
    return mc.get_rank()

def mc_finalize():
    mc.finalize()
# END CRAY ADDED
```

Keras + CPE ML Plugin: Modifications to backend

- **Broadcast Initial Model**
  - A bit complicated for a slide
    - See source example:
      - /scratch/snx3000/kristyn/CUG2018/keras/ tensorflow_backend_ml_comm.py

- **Gradient Aggregation**

```
# CRAY MOD
    grads =tf.gradients(loss,variables,colocate_gradients_with_ops=True)
    grads_mc = mc.gradients(grads, 0)
    return grads_mc
#    return tf.gradients(loss, variables, colocate_gradients_with_ops=True)
# END CRAY MOD
```

Keras + CPE ML Plugin: Changes to user script

- Starting point was keras/examples/mnist_cnn.py
- Import some extra libraries at the beginning

```
# CRAY ADDED
import numpy as np
import os
import math
nnodes = int(os.environ['SLURM_NNODES'])
# END CRAY ADDED
```

- No other changes to user script until after we compile model

```
model.compile(... )
#CRAY ADDED
# calculate input parameters to pass through to backend
K.mc_init( …)
#END CRAY ADDED
model.fit( … )
```

Keras + CPE ML Plugin: Changes to user script

- Need to determine number of trainable parameters, adjust number of epochs, calculate max_steps, then call backend to perform init

```
# CRAY ADDED

# Determine number of trainable variables
trainable_count = int(np.sum([K.count_params(p) for p in set(model.trainable_weights)]))

# Adjust epochs based on parallel throughput
epochs = int(epochs/nnodes)

# Compute max_steps
ntrain_samples = x_train.shape[0]
ntest_samples = x_test.shape[0]
total_steps = int(math.ceil(epochs * (ntrain_samples + ntest_samples)/batch_size))

# threads per team, number of teams, algorithm to use, grads len, total steps, ksteps, v, freq
K.mc_init(1, 1, 0, trainable_count, total_steps, 2000, 2, 1000)

#  END CRAY ADDED
```

Running Keras + CPE ML Plugin on Piz Daint

Load modules

```
$ module load daint-gpu
$ module load TensorFlow/1.3.0-CrayGNU-17.08-cuda-8.0-python3
$ module load craype-ml-plugin-py3/1.0.1
```

Allocate GPU nodes from SLURM

```
$ salloc --nodes=2 --exclusive --constraint=gpu
```

Run

```
srun -n 2 python3 keras-2.1.5/keras/examples/mnist_cnn_ml.py
```

Class Exercise: Keras + CPE ML Plugin

- Install keras
- Run keras mnist_cnn model (or other examples)
- Modify keras tensorflow_backend.py and mnist_cnn model to use CPE ML Plugin
- Apply CPE ML Plugin to other keras example models
  - acgan_mnist model
- Step-by-Step Instructions:
  - /scratch/snx3000/kristyn/CUG2018/Keras/README

Q & A

Charles Siegel, Cray Inc.
csiegel@cray.com

HPC and Analytics with R

Session II

James Maltby, Cray Inc.

What is R?

- **R project for Statistical Computing**
  - https://www.r-project.org
  - Environment for statistical computing and graphics
  - “GNU S”
  - Freely available – but note most R packages have licenses
    - (GPL-2, GPL-3, MIT, Apache, etc.)
  - Latest Version R 3.5.0 (Joy in Playing)
    - R version 3.5.0 (2018-04-23) -- "Joy in Playing"
- **CRAN - The Comprehensive R Archive Network**
  - https://cran.r-project.org
  - Network of ftp and web servers that store identical, up-to-date, versions of code and documentation for R
  - R manuals
    - https://cran.r-project.org/doc/manuals/

Interactivity and R

- **R community was developed with the goal of interactive exploration of data**
  - Basic R interactive console – provided with standard distribution
  - Many R users work use R using an IDE
- **RStudio is by far the most popular IDE for R**
  - R Markdown files and R Notebooks
  - Files have extension .Rmd
- **R can also be run using Jupyter Notebooks**
  - Install IRKernel

What we plan to cover in the tutorial

- **Versions of R on Cray-XC**
  - Cray PE version
  - R provided with Urika-XC
- **Installing R packages**
- **Using R from Jupyter Notebooks**
- **Using Anaconda to manage R packages and multiple R versions (environments)**
- **Setting up a R cluster using “parallel” package**
- **Setting up a pdbR environment (pdbMPI)**

Versions of R provided on XC

- **Cray PE provides R prebuilt with Cray libsci using the GNU compiler**
  - module load cray-R
  - Currently supported version is 3.3.3
- **R is also provided with Urika-XC**
  - Urika-XC 1.1 release
    - Uses prebuilt R from EPEL repository (R version 3.4.2)
      - Installed in /usr/lib64/R in the Urika-XC container
      - Provides R built as a shared/dynamic library
        - Can use R helper routines such a ”littler”
        - Flag --enable-R-shlib causes the make process to build R as a dynamic (shared) library, typically called libR.so, and link the main R executable against that library
        - Possible performance penalty (10%) mentioned in R install notes – have not verified on XC
    - Primarily included to support sparkR
  - Planned for Urika-XC 1.2 release
    - R version 3.5.0 (or most recent release)
    - Plan to provide R prebuilt with Cray libsci + GNU compiler
    - R built as a shared/dynamic library
    - pbdR Ecosytem pre-installed using Cray MPI (initial base set of packages)
    - Support for using Jupyter notebooks via IRKernel

Installing R Packages from CRAN

- **Bring up R on login node and install needed packages**
  - Need external access to download packages
  - In general, most tested, and most reliable compiler for R packages are the GNU compilers (gcc, gfortran)
  - Note, if using a site-installed version, any additional installed packages will be saved to a location in your home directory
    - ~/R/x86_64-suse-linux-gnu-library/3.3
- **R packages we will be using for the tutorial**
  - install.packages("foreach")
  - install.packages("doParallel")
  - install.packages("rlecuyer")
  - install.packages("randomForest")
  - install.packages("SPARQL")

Installing R packages within Urika-XC

- **Use start_analytics**
  - Specify interactive node to run on login node
  - Better connectivity than from XC compute node
- **For Urika-XC 1.1**
  - R version 3.4.2 (2017-09-28) -- "Short Summer"
  - User packages installed to
    - ~/R/x86_64-redhat-linux-gnu-library/3.4

Using R on Piz Daint using Urika-XC

```
kristyn@daint101:~> module load analytics
kristyn@daint101:~> start_analytics –d
```

Once inside the container, bring up R interactive shell to install packages

```
bash-4.2$ R
```

Inside R interactive shell

```
> install.packages("foreach")
```

```
Installing package into ‘/usr/lib64/R/library’
(as ‘lib’ is unspecified)
Warning in install.packages("foreach") :
  'lib = "/usr/lib64/R/library"' is not writable
Would you like to use a personal library
instead?  (y/n) y
Would you like to create a personal library
~/R/x86_64-redhat-linux-gnu-library/3.4
to install packages into?  (y/n) y
```

Running R in Jupyter using Urika-XC

Since Piz Daint is only accessible via ssh from the front end ela.cscs.ch, we need to create a tunnel through the front end to daint to use Web Uis (Jupyter, also for CGE)

Example:

Create tunnel from laptop to internal daint node through ela.cscs.ch

```
$ ssh -L localhost:8022:daint:22 ela.cscs.ch
```

Then in a second terminal, log directly into the daint node

```
$ ssh -p 8022 -L localhost:15000:localhost:15000 localhost
```

Bring up Urika-XC

```
$ start_analytics -d --login-port 8022 --ui-port 15000
```

Then inside container, start Jupyter Notebook

```
bash-4.2$ jupyter notebook --port 15000
```

Managing R using Anaconda

- **Anaconda R**
  - Quite useful for managing R packages and multiple R environments on XC
  - List of R language packages available for install from conda is located at http://repo.continuum.io/pkgs/r/
  - R Essentials bundle includes about 100 of the most popular packages for R

```
> conda create --name myR -c r r-essentials
> source activate myR
```

- **Also can specify specific versions of R**

```
> conda create --name myR_3.2.2 -c r r=3.2.2
```

- **When using an older version of R I found it works better to create the conda environment first, activate this, then install the allowing packages, allowing conda to manage the package version dependencies**

```
> source activate myR_3.2.2
> conda install -c r r-essentails r-xml
```

Enabling ssh between nodes using start_analytics

- **On SLURM-based systems**
  - salloc -N 4 -C mc --image=custom:analytics-1.01.0000.201712122205_0082-latest start_analytics –ssh
- **On interactive node**
  - > # Determine nid allocations
  - > echo “$SLURM_NODELIST” or env | grep SLURM
  - > SLURM_NODELIST=nid0000[4-7]
  - > # Start up R
  - > R

Simple Parallel Socket Cluster

- **Basic functionality**
  - Runs 'Rscript' on the specified host(s) to set up a worker process which listens on a socket for expressions to evaluate, and returns the results (as serialized objects).
- **Commonly used R packages which then build upon the "parallel" package**
  - "foreach" package
    - Provides looping construct
  - "doParallel" package
    - Provides mechanism needed to execute **foreach** loops in parallel
  - https://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

Example datasets

```
> # Base install of R already includes several datasets
> # To look at the datasets available in loaded packages
> data()

> # load the iris dataset
> data(iris)
> head(iris)

> # Many R packages also contain additional datasets
> install.package('rattle')
> data(wine, package='rattle')

> # Also can import data directly
> # Here read.table reads a file in table format and creates a dataframe from it
> url <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'
> whitewine <- read.table(url,header=TRUE,sep=";")
> head(whitewine)
```

Example Code: using foreach and doParallel

```
> library(parallel)
> library(foreach)
> library(doParallel)
> machineVec <- c(rep("nid00004",4), rep("nid00005",4), rep("nid00006",4), rep("nid00007",4))

> cl <- makeCluster(machineVec)
> # To use the "foreach", we need to register the cluster with
> registerDoParallel(cl)
> getDoParWorkers()

> # sequential execution
> system.time(foreach(i=1:100000) %do% sum(tanh(1:i)))
> # parallel execution
> system.time(foreach(i=1:10000) %dopar% sum(tanh(1:i)))

> mcoptions <- list(preschedule=FALSE, set.seed=FALSE, cores=4)
> system.time(foreach(i=1:100000,.options.multicore=mcoptions) %dopar% sum(tanh(1:i)))
```

Example Code: randomForest

```
> # Parallel execution of randomForest
> x <- matrix(runif(500), 100)
> y <- gl(2, 50)
>
> library(randomForest)
>
> rf <- foreach(ntree=rep(25000, 6), .combine=combine,
  .multicombine=TRUE, .packages='randomForest')
  %dopar% { randomForest(x, y, ntree=ntree)}
```

R profiling

- Standard approach – use Rprof
  - Profile R code is to use the Rprof function to profile and the summaryRprof function to summarize the result

```
> help(Rprof)


> Rprof(tmp <- tempfile())
> example(glm)
> Rprof()
> summaryRprof(tmp)
```

Programming with Big Data in R (pbdR)

- Set of highly scalable R packages for distributed computing in data science
  - http://r-pbd.org/
- George Ostrouchov, Wei-Chen Chen, Drew Schmidt, Pragneshkumar Patel
- Winner of the Oak Ridge National Laboratory 2016 Significant Event Award for "Harnessing HPC Capability at OLCF with the R Language for Deep Data Science

Installing pdbMPI package

- **If not already installed, install rlecuyer package**
  - wget https://cran.r-project.org/src/contrib/rlecuyer_0.3-4.tar.gz
  - R CMD INSTALL --no-test-load rlecuyer_0.3-4.tar.gz
- **Install pdbMPI package**
  - wget https://cran.r-project.org/src/contrib/pbdMPI_0.3-3.tar.gz
  - R CMD INSTALL pbdMPI_0.3-3.tar.gz --configure-args="--with-mpi=/opt/cray/pe/mpt/default/gni/mpich-gnu/51/ --disable-opa --with-mpi-type=MPICH2" --no-test-load
- **https://cran.r-project.org/web/packages/pbdMPI/pbdMPI.pdf**

pdbMPI: run “Hello World”

Create file mpi_hello_world.r

```
# load the package
suppressMessages(library(pbdMPI, quietly = TRUE))

# initialize the MPI communicators
init()

# Hello world
message <- paste("Hello from rank", comm.rank(), "of", comm.size())
comm.print(message, all.rank=TRUE, quiet=TRUE)

# shut down the communicators and exit
finalize()
```

```
> srun -N 4 Rscript mpi_hello_world.r
```

pbdMPi – beyond “Hello World”

- HPSC Cookbook – Wei-Chen Chen
- https://snoweye.github.io/hpsc/cookbook.html

- In addition there are several tutorials available with source code available for download
- Tutorials 1 and 2 both use the Iris dataset already available with base R install

Parallel (SPMD) pi Example (from HPSC)

```
# File name: ex_pi_spmd.r
# Run: srun -N 2 Rscript --vanilla ex_pi_spmd.r

### Load pbdMPI and initial the communicator.
library(pbdMPI, quiet = TRUE)
init()
.comm.size <- comm.size()
.comm.rank <- comm.rank()

### Compute pi.
n <- 1000
totalcpu <- .comm.size
id <- .comm.rank + 1
mypi <- 4*sum(1/(1+((seq(id,n,totalcpu)-.5)/n)^2))/n    # The example from Rmpi.
mypi <- reduce(mypi, op = "sum")

### Output from RANK 0 since mpi.reduce(...) will dump only to 0 by default.
comm.print(mypi)
finalize()
```

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publicly announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA and YARCDATA. The following are trademarks of Cray Inc.: CHAPEL, CLUSTER CONNECT, CLUSTERSTOR, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE, REVEAL. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used on this website are the property of their respective owners.

Q & A

James Maltby, Cray Inc.
jmaltby@cray.com