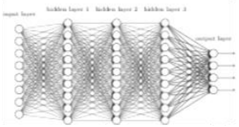
Scaling DL Training Workloads with the Cray PE Plugin

Luiz DeRose
Sr. Principal Engineer
Programming Environments Director

Autonomous Driving



Training data

Results

CNN Training

Labeled Data

Supervised Learning

Model

Backend

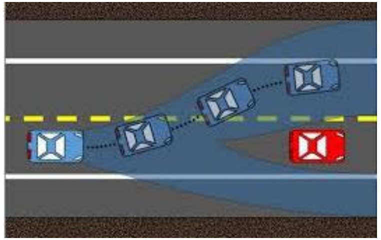
Frontend

Real-time data → **Inference** → **Mission and Trajectory Planning**

Sensors

Sensor fusing

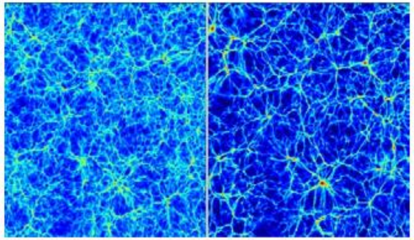
Perception | Cognition | Action



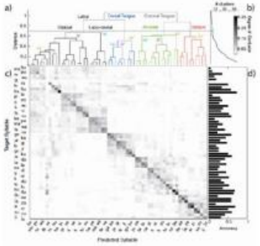
Deep Learning in Science - NERSC

Opportunities to apply DL widely in support of classic HPC simulation and modelling.

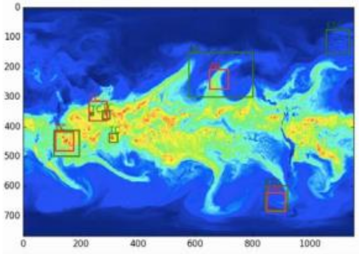
Modeling galaxy shapes



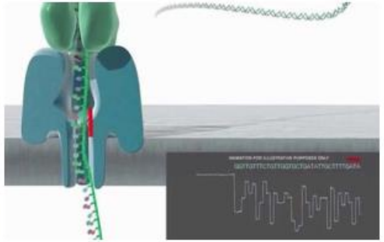
Generating cosmology mass maps



Decoding speech from ECoG



Detecting weather patterns in simulation and satellite datasets



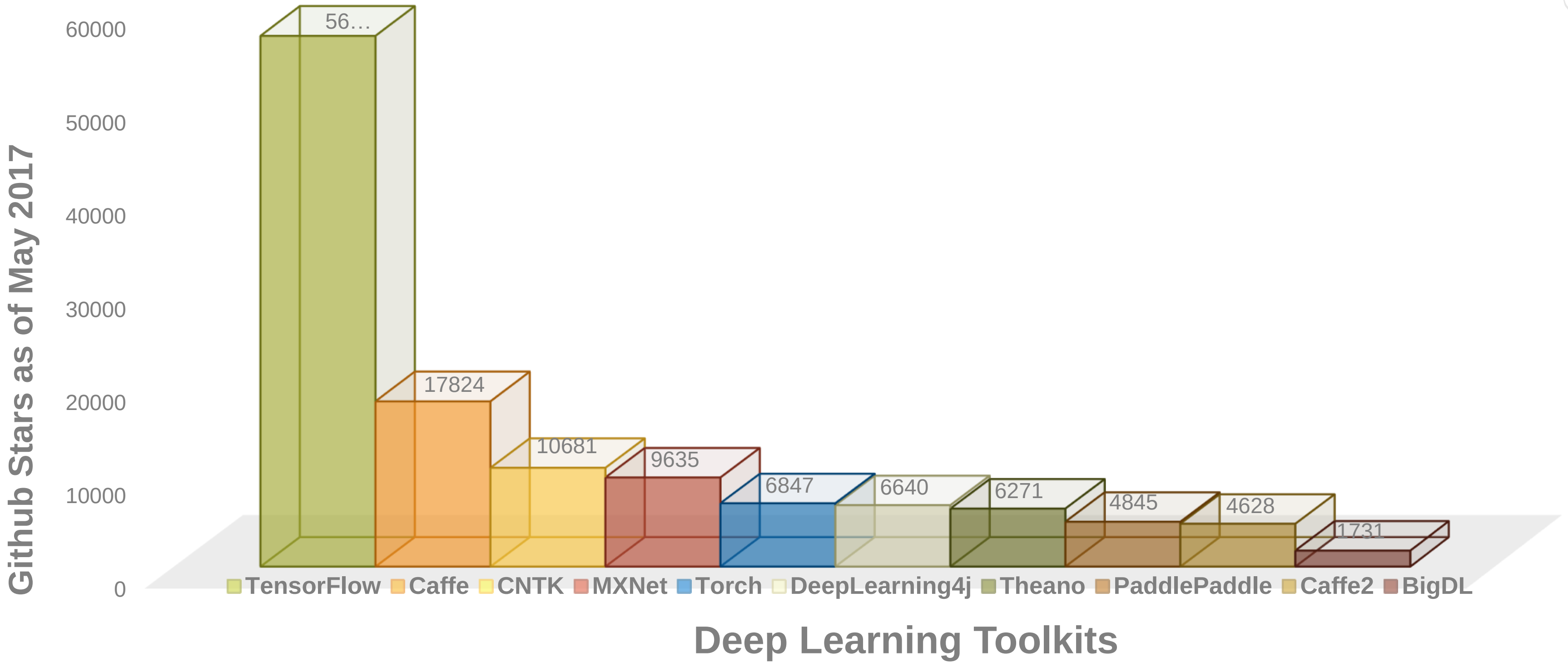
Oxford Nanopore sequencing

The Deep Learning Part in Autonomous Driving

- **Model training is the most crucial and challenging aspect**
  - Many tasks to train for in autonomous driving
    - **Stereo**
    - **Optical flow**
    - **Visual odometry**
    - **Structure-from-motion**
    - **Object detection**
    - **Recognition and tracking**
    - **3D scene understanding**
  - Many NNs to train for each task
  - Retraining (or transfer learning) happens when new data is available

- **A trained neural network can also be a powerful tool for**
  - Pattern recognition
  - Classification
  - Clustering
  - Others…

The Deep Learning Software Landscape



TensorFlow is nearly as popular as all other frameworks combined

Scaling Deep Learning - Motivation

- **Scaling Deep Learning training is a tool for**
  - Models that take a very long time to train
    - and have a very large training dataset
  - Increasing the frequency at which models can be retrained with new or improved data

- **It is critical to be able to update the models (when new data arrives) in a matter of minutes**

- **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

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

The Cray PE DL Scalability Plugin Project Overview

- **Cray's primary goals were:**
  - Design a solution for scaling TensorFlow (specifically synchronous SGD) to significantly larger node counts than existing methods allowed
    - Should require **minimal changes to user training scripts** and provide a **more friendly user experience**
  - Achieve the **best possible TensorFlow performance on Cray Systems**
  - **Maintain accuracy** for a given number of steps and hyper-parameter setup allowing for significantly reduced time-to-accuracy through scaling
  - Ideally have a **portable solution** that would work with other deep learning frameworks

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**

$\quad G_{local} \leftarrow \text{COMPUTE\_GRADIENTS}(\text{mini batch})$

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

$\quad \text{APPLY\_GRADIENTS}(G_{global})$

end for

Compute intensive

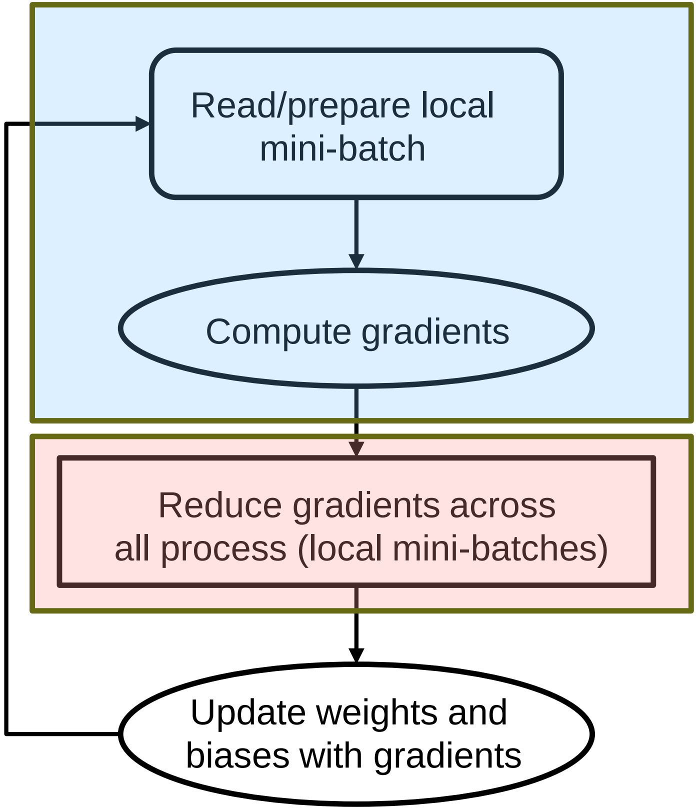
Communication intensive

Typically not much compute

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

Data Parallel Synchronous SGD

- **Operations in a "step" of SGD**
- **Parallel operations highlighted with light blue box**
- **Communication highlighted in light red box**
  - Maps to an allreduce
- **Non-parallel work is the remainder**
- **For a fixed local mini-batch size per process, the fraction of time in parallel work is constant as more processes are added**



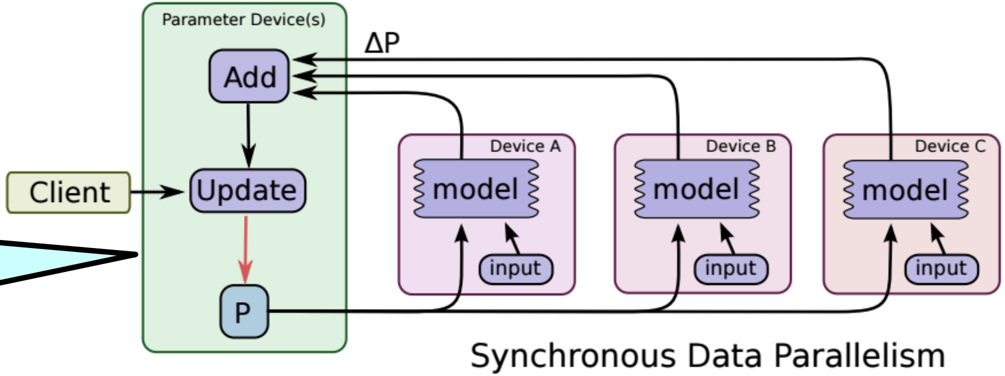
Distributed TensorFlow

- **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

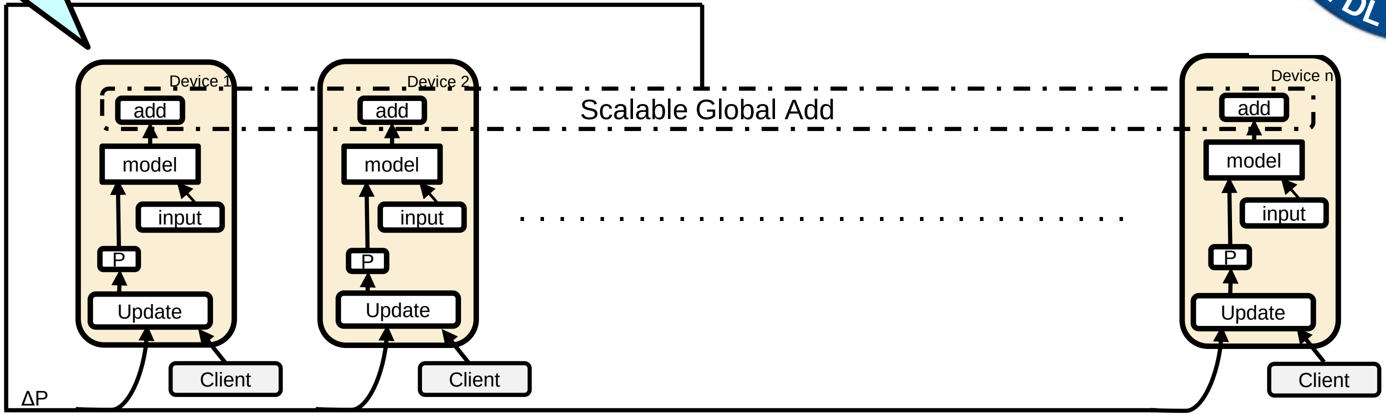
Distributed TensorFlow

No Parameter devices needed in the Cray method

Resources dedicated to gradient calculation



Cray Method

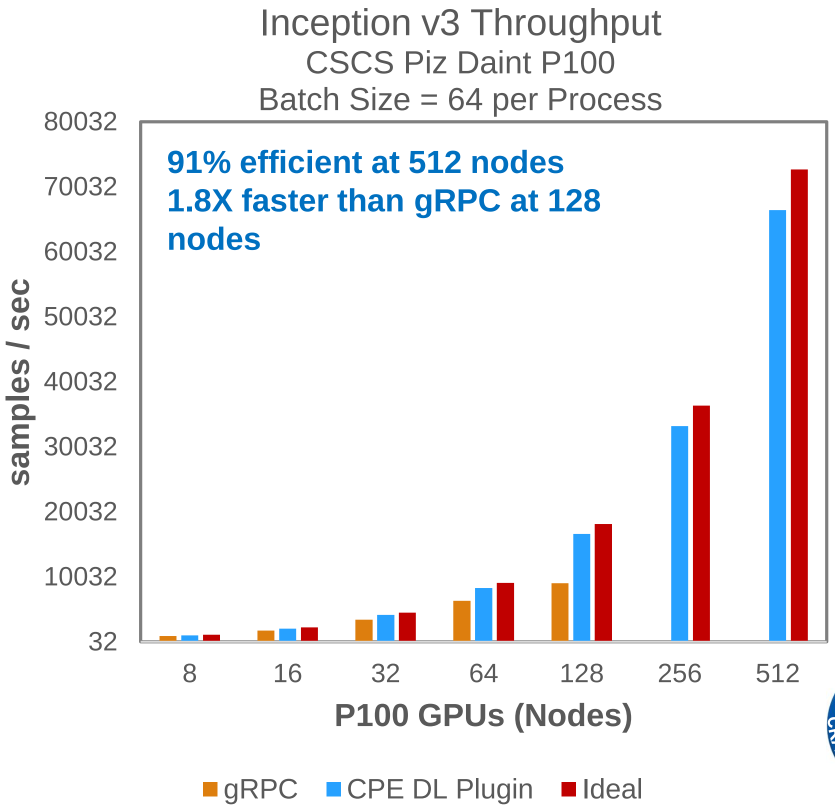


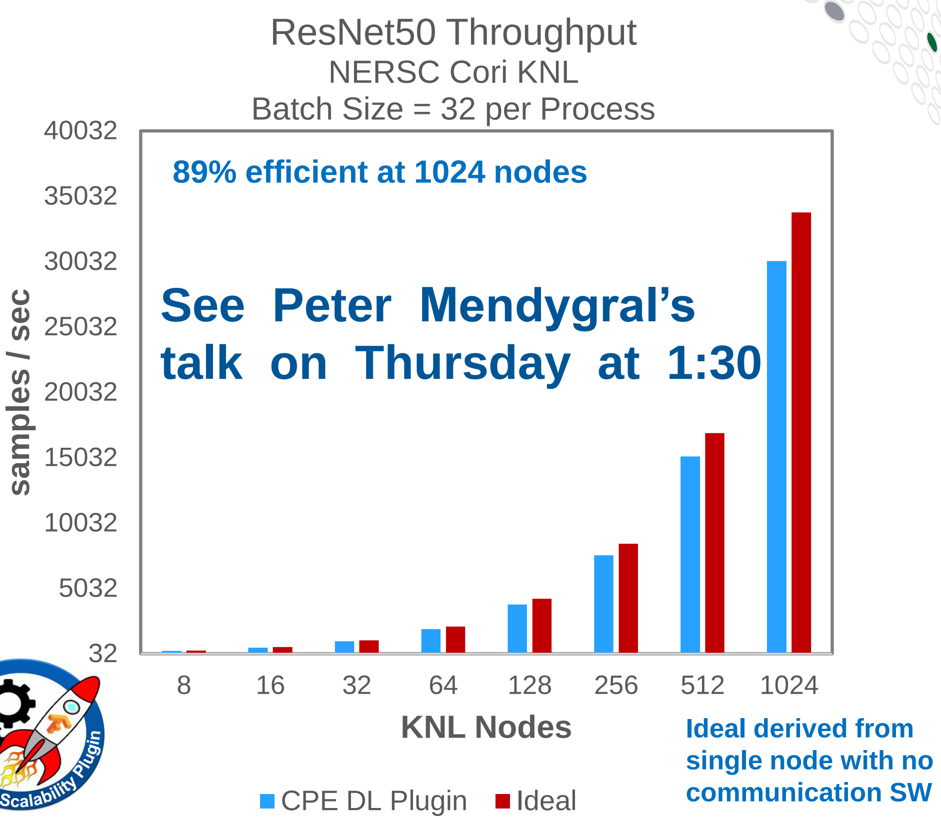
Training Script Modifications

- **The Cray PE DL Plugin require the following modifications to a serial training script**

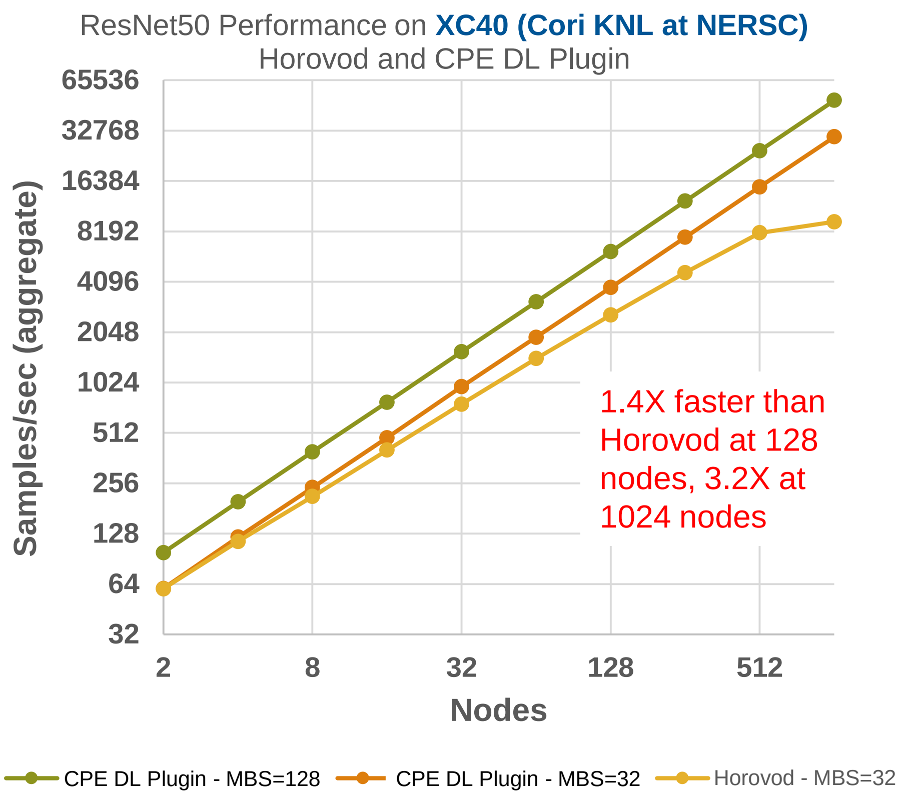
1. Importing the Python module
2. Initialize the module
   - Possibly configure the thread team(s) for specific uses
3. Broadcast initial model parameters
4. Incorporate gradient aggregation between gradient computation and model update
5. Finalize the Python module

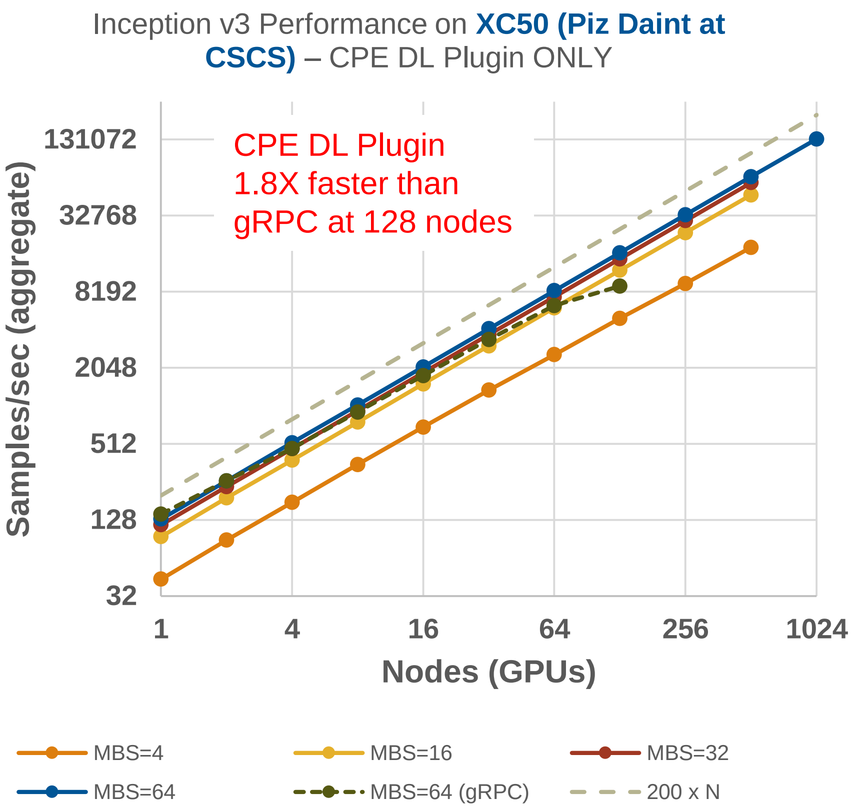
Cray PE DL Scalability Plugin Performance





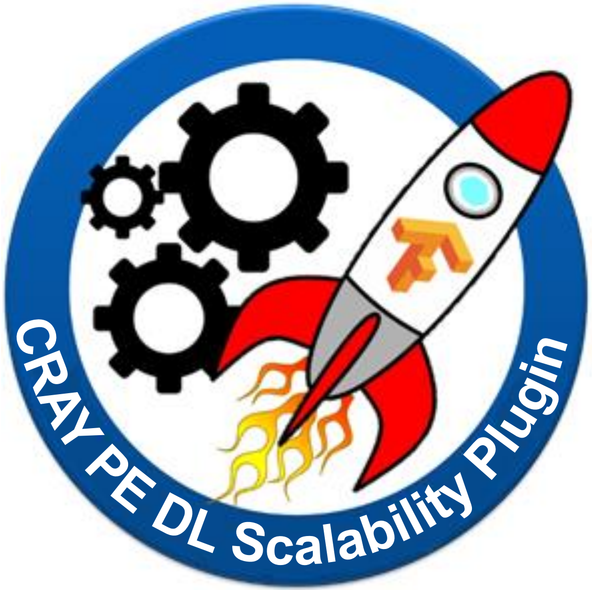
Horovod / CPE DL Plugin – Throughput Scaling





Cray PE DL Scalability Plugin

- **Users can easily achieve ideal scaling performance across DL frameworks utilizing stochastic gradient descent**
  1. Load a module
  2. Plug in a few simple lines to your serial Python or C-based training script
  3. Scale up your training workload to hundreds of nodes or more on Cray systems
- **Delivers high performance across a variety of Cray node architectures**
  - Cray customers can leverage their existing compute nodes to scale DL training



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.

COMPUTE | STORE | ANALYZE

Questions?

Thank You!