VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Efficient multi-GPU and multi-node execution of Deep and Machine Learning frameworks

Georg Zitzlsberger georg.zitzlsberger@vsb.cz

03-11-2021

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research, Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS

Agenda

Introduction to Data Parallel Deep Learning with Horovod

- Parallelism
- Horovod

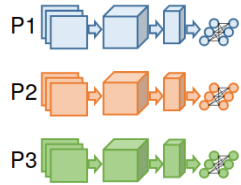
Multi-node/-GPU aware Data Processing Pipelines

- Data Pipeline with Tensorflow 2.0 and Keras
- Tensorflow Dataset Recommendations

Demonstration of Multi-node/-GPU Examples using Tensorflow Multi-node/-GPU Machine Learning with scikit-learn

Introduction to Data Parallel Deep Learning with Horovod

Difference Data vs. Model Parallelism

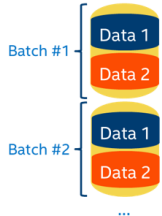


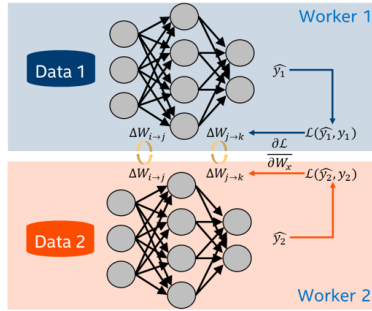
- Network layers assigned to different workers
- Every worker trains with the same data
- Activations are exchanged (requires large I/O bandwidth)
- **Enables bigger models**

- All workers see the same network
- Every worker trians with different data
- Gradints (weights) are exchanged (averaging to common model)
- Side effect: "sharp" minima
- **Enables faster training**

(Images: Ben-Nun, et al.)

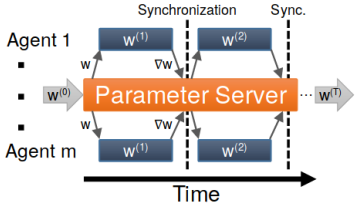
Distributed Training: Data Parallelism in Detail



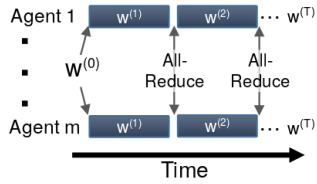


- Batch size limits parallelism
- Scaling batch size requires scaling of learning rate (linearly)

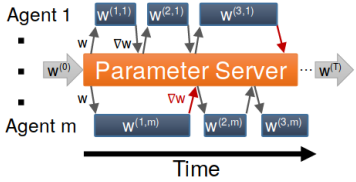
Distributed Training: Model Consistency



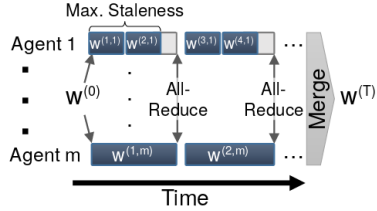
(a) Synchronous, Parameter Server



(b) Synchronous, Decentralized



(c) Asynchronous, Parameter Server



(d) Stale-Synchronous, Decentralized

(Image: Ben-Nun, et al.)

Horovod

- Developed by Uber Engineering
- Part of Michelangelo (Uber's Machine Learning Platform)
- Aimed at and demonstrated for large scale
- Uses MPI based collective communication (synchronous & decentralized)
- Only small code modifications needed
- Supports the most common frameworks:
  - Tensorflow (1.x & 2.0) + Keras
  - Pytorch
  - MXNet

How to Use Horovod

- Horovod comes with a wrapper horovodrun, e.g.:
  `$ horovodrun -np 4 -H server1:2,server2:2 python train.py`
- Different back-ends are possible: MPI, Gloo, NCCL, oneCCL, etc.
- Intel MPI or OpenMPI can be used:
  `$ mpirun -n 4 -ppn 2 -hosts server1,server2 python train.py`

Horovod for KERAS/Tensorflow 2.0

- Add the following:
  - `hvd.init()`:
    Initializes Horovod (and MPI underneath)
  - `hvd.callbacks.BroadcastGlobalVariablesCallback(0)`:
    Initialize model to start with same copies
  - `hvd.DistributedOptimizer(...)`:
    Wrapper around standard optimizer (SGD, Adam, etc.) to enable distributed weight/gradient updates

- **Note: The same script is executed on all workers!**
  Only let first rank do the I/O (e.g. print to stdout or save snapshots)

- Full documentation can be found here

Horovod for KERAS/Tensorflow 2.0

What needs attention:

- If `tf.data.Dataset` is used, consider `shard(num_shards, index)`, e.g.: `my_dataset.shard(hvd.size(), hvd.rank())`
- If training steps are used, instead of number of epochs, adjust the steps, e.g.: `training_steps /= hvd.size()` (assuming perfectly balanced training data)
- If training data size is large, avoid loading it at every worker and divide accross workers
  The same script is executed on all workers!
- Scale the learning rate linearly with the number of workers, e.g.:
  `lr *= hvd.size()`
  See Alex Krizhevsky's paper:
  Strictly speaking it should be `lr *= sqrt(hvd.size())`

A Few Words on Native Data Parallel Training

Native support of (sync.) data parallel training is also available:

- Tensorflow:
  `tf.distribute.Strategy` with different strategies (MirroredStrategy, TPUStrategy, MultiWorkerMirroredStrategy, CentralStorageStrategy, ParameterServerStrategy)
  ▸ Documentation

- Pytorch:
  `torch.distributed` with three backends (GLOO, MPI, NCCL)
  ▸ Documentation

Multi-node/-GPU aware Data Processing Pipelines

Data Pipeline with Tensorflow 2.0 and Keras

- Extract, Transform and Load (ETL) pipeline via `tf.data.Dataset`
- Provides a wide range of functionality to process training/validation data:
  - I/O: files, NumPy, TFRecord/Protocol Buffers, Pandas Data Frames, etc.
  - Split training/validation:
    Provide a ratio how much of the dataset should be for training.
  - Batch and pad:
    Build minibatches and pad to ensure balance.
  - Shuffle:
    Randomize the samples with every training epoch.
  - Cache and Pre-fetch:
    Optimize access to data.
  - Map and filter:
    Convert the data to a format needed for training/validation and also filter samples.
  - ...

Data Pipeline: Example

- Example `keras_mnist_example_1.ipynb`, uses the following training pipeline:
  - Input MNIST training dataset `ds_train` and apply `normalize_img` with `tf.data.experimental.AUTOTUNE` parallel calls, via
    - `tf.data.Dataset.map`
  - Cache the data (no repeated normalization) with
    - `tf.data.Dataset.cache`
  - Suffle data entirely (size of `ds_info.splits['train'].num_examples`) with
    - `tf.data.Dataset.shuffle`
  - Batch with a batch size of 128 with
    - `tf.data.Dataset.batch`
  - Prefetch the next elements (use `tf.data.experimental.AUTOTUNE` for the buffer size)
    - `tf.data.Dataset.prefetch`
- The validation pipeline `ds_test`, does the same **except** suffling

See the Tensorflow Dataset Documentation for more information

Tensorflow Dataset Recommendations

- Some methods offer multi-threading; try `tf.data.experimental.AUTOTUNE`, e.g.:

```
train_ds = tf.data.Dataset.from_tensor_slices(my_data)
                    .map(my_prepare_func, num_parallel_calls=AUTO))
```

- Caching is keeping everything in memory - be carefull where to place it in the pipeline!
- Caching can also be used to use fast NVM/SSD storage, e.g.:
  `train_ds.cache(filename="/mnt/nvmeof/train_ds_{}".format(hvd.rank()))`
- Use `tf.data.Dataset.map` before `tf.data.Dataset.batch` if map is expensive, vice versa otherwise
- Prefetch at the end of the pipeline

See Tensorflow's ▸ Better performance with the tf.data API

Caching

- The *SCRATCH* filesystem is used for projects
- Reloading training/validation data from *SCRATCH* is not efficient:
  - No guaranteed I/O bandwidth
  - Hogging of resources
- Solution: Cache dataset pipelines using
  - Ramdisk (global with `qsub ...-l global_ramdisk=true`)
  - NVMeoF (Non-Volatile Memory express over Fabric)
- Barbora:
  - Ramdisk with 180GB of 192GB per node on `/mnt/global_ramdisk/`
  - NVMeoF shared on nodes with `qsub ...-l nvmeof=700GB:shared` on `/mnt/nvmeof/` (max. 10TB)
- Karolina:
  - Ramdisk with approx. 1TB per node (`qnvidia`) on `/mnt/global_ramdisk/`
  - ~~NVMeoF~~ (not yet)

Demonstration of Multi-node/-GPU Examples using Tensorflow

Multi-node/-GPU Machine Learning with scikit-learn

CPU Optimizations

- Intel offers own version of scikit-learn via Intel Distribution for Python
- It builds on Intel's performance libraries:
  - Intel Data Analytics Acceleration Library (Intel DAAL):
    Replaces some algorithms/tools from scikit-learn
  - Intel Math Kernel Library (Intel MKL):
    Backend to NumPy and SciPy
- Intel DAAL substitutions are turned off by default, enable with:

```
import daal4py.sklearn
daal4py.sklearn.patch_sklearn()
```

. . . or

```
$ USE_DAAL4PY_SKLEARN=YES python ...
```

- Don't use toy sets to test speedups; there is some overhead involved

GPU Optimizations

- From the scikit-learn FAQ:
  Q: Will you add GPU support?
  A: No, or at least not in the near future. ...
- Some tools/algorithms are available by H2O4GPU:

```
#from sklearn.cluster import KMeans
from h2o4gpu.solvers import KMeans
```

or

```
import h2o4gpu as sklearn
```

⇒ It can be used as drop-in replacement

- *H2O4GPU*:
  - Inherits *scikit-learn* tools/algorithms
  - Adds GPU support to some; falls back to CPU if not available
  - Builds on existing GPU solvers

H2O4GPU

Advantages:

- Selected algorithms can significantly benefit from higher GPU throughput:
  - GLM: Lasso, Ridge Regression, Logistic Regression, Elastic Net Regulariation
  - KMeans
  - Gradient Boosting Machine (GBM) via XGBoost
  - Singular Value Decomposition(SVD) + Truncated Singular Value Decomposition
  - Principal Components Analysis(PCA)
- Transparently falls back to scikit-learn implementation (Intel DAAL)
- Supports multiple GPUs
- Offers more tools than scikit-learn

Disadvantages:

- Data transfers between host and GPU are limiting the speedup (or even slow down)
- Lacks behind scikit-learn and has smaller community

H2O4GPU Example

```
import matplotlib
from h2o4gpu import DAAL_SUPPORTED
from sklearn import datasets, linear_model
import matplotlib.pyplot as plt
import numpy as np

diabetes = datasets.load_diabetes()
# Define diabetes_X_train, diabetes_X_test, diabetes_y_train, diabetes_y_test
...

if DAAL_SUPPORTED:
    from h2o4gpu.solvers.daal_solver.daal_data import getNumpyShape
    import h2o4gpu

    lin_solver_daal = h2o4gpu.LinearRegression(fit_intercept=True,
                                               verbose=True,
                                               backend='daal',
                                               method=h2o4gpu.LinearMethod.normal_equation)

    rows, cols = getNumpyShape(diabetes_y_train)
    y = diabetes_y_train.reshape(cols, rows)
    lin_solver_daal.fit(diabetes_X_train, y)
    daal_predicted = lin_solver_daal.predict(diabetes_X_test)
else:
    from sklearn.metrics import mean_squared_error, r2_score

    # Create linear regression object
    regr = linear_model.LinearRegression()

    # Train the model using the training sets
    regr.fit(diabetes_X_train, diabetes_y_train)

    # Make predictions using the testing set
    diabetes_y_pred = regr.predict(diabetes_X_test)
```

Some Notes on Use

- Some algorithms/tools from scikit-learn exist in optimized versions for CPUs and GPUs
- They require sufficient workload/minimized data transfer to benefit
- H2O4GPU and Intel DAAL offer even more algorithms but with their own API
- **Be aware:**
  They are different implementations and can lead to different results[1]!

[1] If bitwise reproducibility is required.

Singularity

- Container system for HPC
- Convert a Docker container to a Singularity image:
  - docker2singularity
- Example:

```
$ module load Singularity/3.8.0
$ module load CUDA/11.0.2-GCC-9.3.0
$ singularity exec --nv -B /scratch/project/open-21-31:/work ↙
      my_container.sif jupyter lab --port 8888
```

- Get ready-to-use images from the NVIDIA GPU Cloud

More to Learn

- IT4Innovations is an NVIDIA Deep Learning Institute

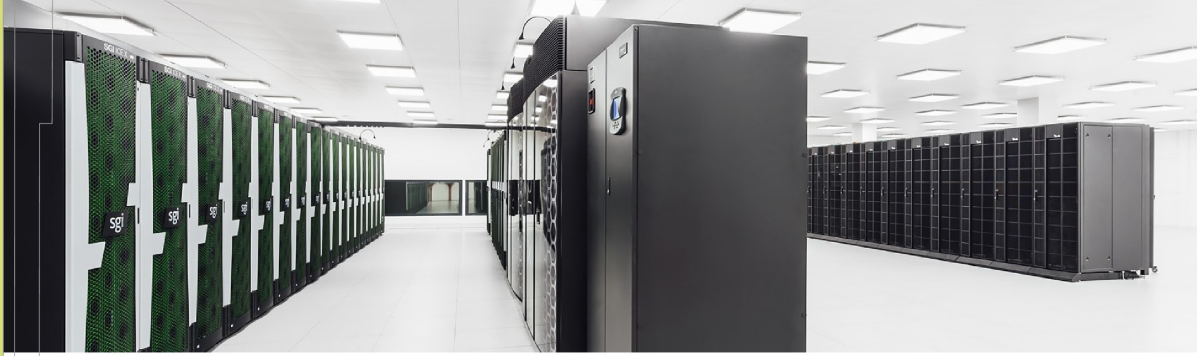
VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

NVIDIA DEEP LEARNING INSTITUTE UNIVERSITY AMBASSADOR

- We offer different instructor-led courses:
  - Fundamentals of Deep Learning (early 2022)
  - Fundamentals of Deep Learning for Multi-GPUs
  - Building Transformer-Based Natural Language Processing Applications (early 2022)
  - ~~Fundamentals of Deep Learning for Computer Vision~~ (EOL)
  - ~~Fundamentals of Deep Learning for Multiple Data Types~~ (EOL)
- And more: find our offering of training courses

PRACE EURO

- Contact us on training@it4i.cz for on-demand courses



VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

IT4Innovations National Supercomputing Center

VŠB – Technical University of Ostrava
Studentská 6231/1B
708 00 Ostrava-Poruba, Czech Republic
www.it4i.cz

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION, YOUTH AND SPORTS