VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

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

## . . . on LUMI

Georg Zitzlsberger georg.zitzlsberger@vsb.cz

14-02-2023

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

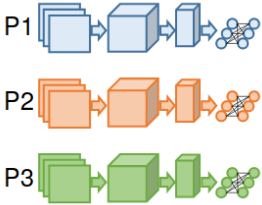
- Data Pipeline with Tensorflow 2.0 and Keras
- Tensorflow Dataset Recommendations

Tensorflow (and PyTorch) on LUMI

Demonstration of Multi-node/-GPU Example using Tensorflow

# Introduction to Data Parallel Deep Learning with Horovod

# Difference Data vs. Model Parallelism

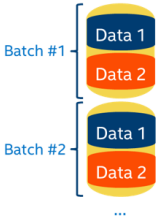



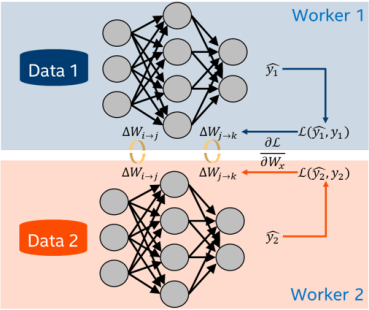

- Network layers assigned to different workers
- Every worker trains with the same data
- Activations are exchanged (requires large I/O bandwidth)
- **Enables bigger models**

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

(Images: Ben-Nun, et al.)

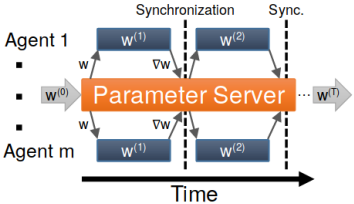
# Distributed Training: Data Parallelism in Detail



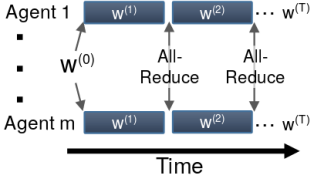




- Batch size limits parallelism (effective batch size)
- Scaling batch size requires (linear) scaling of learning rate

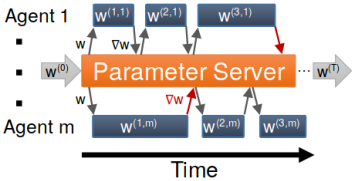
# Distributed Training: Model Consistency



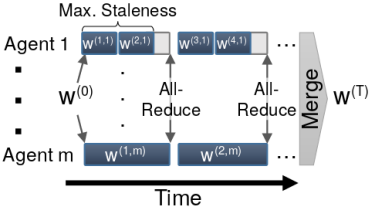

(a) Synchronous, Parameter Server




(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.
- NCCL: NVIDIA Collective Communications Library
- For AMD GPUs: NCCL := RCCL (ROCm Communication Collectives Library)
- 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 across 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

  For LUMI, use ▸ `torch.nn.parallel.DistributedDataParallel` (DDP) with **NCCL** backend

# 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

- In the demonstration, we use 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`
  - Suffle data entirely (size of `ds_info.splits['train'].num_examples`) with
    `tf.data.Dataset.shuffle`
  - Batch with an effective batch size of 64 with
    `tf.data.Dataset.batch`
  - Cache the data (no repeated normalization) with
    `tf.data.Dataset.cache`
  - 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 careful 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

| | Path | Intended use | Hardware partition used |
|---|---|---|---|
| **User home** | `/users/<username>` | User home directory for personal and configuration files | LUMI-P |
| **Project persistent** | `/project /<project>` | Project home directory for shared project files | LUMI-P |
| **Project scratch** | `/scratch /<project>` | Temporary storage for input, output or checkpoint data | LUMI-P |
| **Project flash** | `/flash/<project>` | High performance temporary storage for input and output data | LUMI-F |

| | Quota | Max files | Expandable | Backup | Retention |
|---|---|---|---|---|---|
| **User home** | 20 GB | 100k | No | Yes | User lifetime |
| **Project persistent** | 50 GB | 100k | Yes, up to 500GB | No | Project lifetime |
| **Project scratch** | 50 TB | 2000k | Yes, up to 500TB | No | 90 days |
| **Project fast** | 2 TB | 1000k | Yes, up to 100TB | No | 30 days |

Use the *project scratch* or add to your project the *project flash* if needed.

# Caching - cont'd

- Two options for keeping your training/validation data:
  - *project scratch*: Good I/O
  - *project flash*: Fast I/O but extra project costs
- LUMI-G: 512 GB Memory per node (+4x 128GB per GPU)
- Control caching:
  - If data (and model) fits into CPU memory:
    `train_ds.cache()` is sufficient
  - If CPU memory is not enough:
    `train_ds.cache(filename="/scratch/.../train_ds_{}".format(hvd.rank()))`
    or
    `train_ds.cache(filename="/flash/.../train_ds_{}".format(hvd.rank()))`

# Tensorflow (and PyTorch) on LUMI

# PyTorch on LUMI

LUMI **Documentation**

Search

Intro | First steps | Hardware | Run jobs | Software | Developing | Help desk

**Software**
Overview

**Installing software**
EasyBuild
Spack
Container wrapper

**Containers**
Singularity/Apptainer containers

**Software guides**
PyTorch
Software library
ParaView
QuantumESPRESSO
VASP

**Local software collections**
CSC

## PyTorch on LUMI

PyTorch is an open source Python package that provides tensor computation, like NumPy, with GPU acceleration and deep neural networks built on a tape-based autograd system.

PyTorch can be installed by the users following the code's instructions. The options to choose for LUMI in the interactive instructions are `Linux`, `Pip` and `ROCm5.X`. For installing with `pip`, the `cray-python` module should be loaded. PyTorch comes with ROCm binaries needed for the GPU support. Even if a particular version of ROCm is not available on LUMI, PyTorch may still be able to use the GPUs.

PyTorch can be run within containers as well. In particular, containers from the images provided by AMD on DockerHub. Those images are updated frequently and make it possible to try PyTorch with recent ROCm versions. Another point in favor of using containers, is that PyTorch's installation directory can be quite large both in terms of storage size and number of files.

### Running PyTorch within containers

We recommend using container images from `rocm/pytorch` or `rocm/deepspeed`.

The images can be fetched with singularity:

```
SINGULARITY_TMPDIR=$SCRATCH/tmp-singularity singularity pull docke
```

**Table of contents**



The guide can be found on the ▸ LUMI Software Documentation

# Tensorflow on LUMI

- Install OpenMPI:

```
$ module load LUMI/22.08 partition/G
$ module load EasyBuild-user
$ eb OpenMPI-4.1.3-cpeGNU-22.08.eb -r
```

- Install the `aws-ofi-rccl` plugin:

```
$ eb aws-ofi-rccl-66b3b31-cpeGNU-22.08.eb -r
```

- Install `singularity-bindings`:

```
$ eb singularity-bindings-system-cpeGNU-22.08.eb -r
```

- Get the ROCm enabled Tensorflow container from Dockerhub:

```
$ singularity pull tensorflow.sif
                   docker://rocm/tensorflow:rocm5.4.1-tf2.10-dev
```

# Tensorflow on LUMI (Temporary)

There's a bug in current `rocm5.4.1-tf2.10-dev` that SEGVs Horovod. Let's fix it...

- Install `ensurepip` with the following script (`download-ensurepip.sh`):

```
base_url=https://raw.githubusercontent.com/python/cpython/3.9/Lib

files=("__init__.py" \
       "__main__.py" \
       "_uninstall.py" \
       "_bundled/__init__.py" \
       "_bundled/pip-22.0.4-py3-none-any.whl" \
       "_bundled/setuptools-58.1.0-py3-none-any.whl")

for _f in ${files[@]}; do

    f=ensurepip/${_f}

    if test ! -f "$f"; then
        wget -q --show-progress ${base_url}/${f} -P $(dirname $f);
    else
        echo -e "'${f}' already exists. Nothing to do."
    fi
done

$ bash download-ensurepip.sh
```

- Create a virtual environment:

```
$ singularity exec tensorflow.sif -B $HOME/ensurepip:/usr/lib/python3.9/ensurepip bash
<singularity> python -m venv horovod_env --system-site-packages
```

# Tensorflow on LUMI (Temporary) - cont'd

- Build and install Horovod:

```
$ singularity exec tensorflow.sif bash
<singularity> . horovod_env/bin/activate

<singularity> export HOROVOD_WITHOUT_MXNET=1
<singularity> export HOROVOD_WITHOUT_PYTORCH=1
<singularity> export HOROVOD_GPU=ROCM
<singularity> export HOROVOD_GPU_OPERATIONS=NCCL
<singularity> export HOROVOD_WITHOUT_GLOO=1
<singularity> export HOROVOD_WITH_TENSORFLOW=1
<singularity> export HOROVOD_ROCM_PATH=/opt/rocm
<singularity> export HOROVOD_RCCL_HOME=/opt/rocm/rccl
<singularity> export RCCL_INCLUDE_DIRS=/opt/rocm/rccl/include
<singularity> export HOROVOD_RCCL_LIB=/opt/rocm/rccl/lib
<singularity> export HOROVOD_MPICXX_SHOW="CC --cray-print-opts=all"
<singularity> export HCC_AMDGPU_TARGET=gfx90a

<singularity> pip install gast==0.3.3 numpy==1.22 protobuf==3.19
<singularity> pip install --no-cache-dir --force-reinstall horovod==0.26.1
```

- **Important:**
  Do not load `aws-ofi-rccl` and/or `singularity-bindings` modules when building Horovod!

# Run Tensorflow on LUMI

- Create `run_job.sh`:

```
#!/bin/bash
#SBATCH --job-name=lumi_access
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=8
#SBATCH --time=01:00:00
#SBATCH --partition standard-g
#SBATCH --account=project_XXXXXXXXX
#SBATCH --gpus-per-node=8

module load LUMI/22.08
module load partition/G
module load singularity-bindings
module load aws-ofi-rccl
module load OpenMPI/4.1.3-cpeGNU-22.08

export SCRATCH=/scratch/project_XXXXXXXXX/
export NCCL_DEBUG=INFO
export NCCL_SOCKET_IFNAME=hsn
export MIOPEN_USER_DB_PATH=/tmp/${USER}-miopen-cache-${SLURM_JOB_ID}
export MIOPEN_CUSTOM_CACHE_DIR=${MIOPEN_USER_DB_PATH}
export CXI_FORK_SAFE=1
export CXI_FORK_SAFE_HP=1
export FI_CXI_DISABLE_CQ_HUGETLB=1
export SINGULARITYENV_LD_LIBRARY_PATH=/openmpi/lib:/opt/rocm-5.4.1/lib:${EBROOTAWSMINOFIMINRCCL}/lib:/opt/cray/xpmem
      ↪/2.4.4-2.3_9.1__gff0e1d9.shasta/lib64:$SINGULARITYENV_LD_LIBRARY_PATH

mpirun -np 8 singularity exec -B"$SCRATCH:/work" $SCRATCH/lumi_access/tensorflow.sif bash -c ". horovod_env/bin/
      ↪activate; cd /work/lumi_access/; python keras_horovod_example.py"
```

- Run with: `$ sbatch ./run_job.sh`

# Demonstration of Multi-node/-GPU Example using Tensorflow

# 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
  - Building Transformer-Based Natural Language Processing Applications
  - Applications of AI for Predictive Maintenance
  - **Applications of AI for Anomaly Detection (28-02-2023)**
  - **Data Parallelism: How to Train Deep Learning Models on Multiple GPUs (mid 2023)**
  - ~~Fundamentals of Deep Learning for Computer Vision~~ (EOL)
  - ~~Fundamentals of Deep Learning for Multiple Data Types~~ (EOL)
  - ~~Fundamentals of Deep Learning for Multi-GPUs~~ (EOL)
- Find all our training events at https://events.it4i.cz/

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