Multi-GPU training of deep learning models on Piz Daint

Synchronous Distributed Training with Data Parallelism

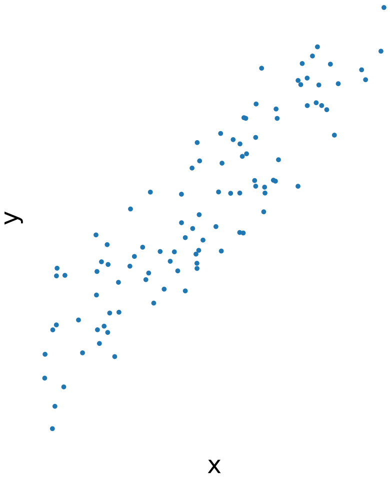
Rafael Sarmiento
ETHZürich / CSCS
CSCS/USI Summer University 2022

ETH zürich
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Outline

- Stochastic Gradient Descent
- [lab] Simple SGD with TensorFlow
- Synchronous Distributed SGD with data parallelism
- Ring Allreduce Algorithm
- [lab] Simple Distributed SGD with TensorFlow and Horovod

We want to train a model on this data



We choose a model and a cost function

$$y = mx + n$$

$$L = \frac{1}{N}\sum_{i}^{N}(\hat{y}_i - y_i)^2$$

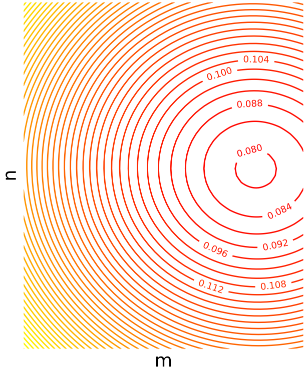
We choose a model and a cost function

$$y = mx + n$$

$$L = \frac{1}{N}\sum_{i}^{N}(\hat{y}_i - y_i)^2$$

$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

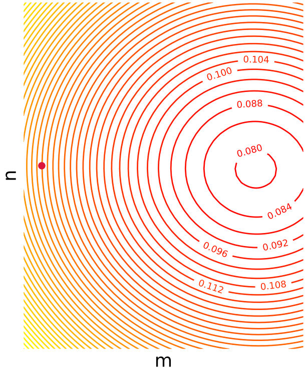
We choose a model and a cost function

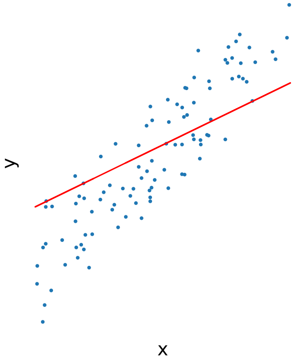


$$y = mx + n$$

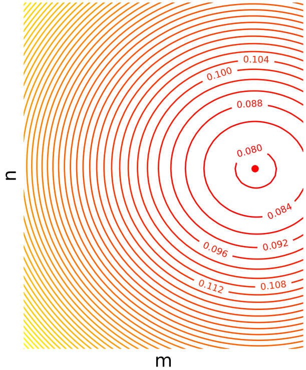
$$L = \frac{1}{N}\sum_{i}^{N}(mx_i + n - y_i)^2$$

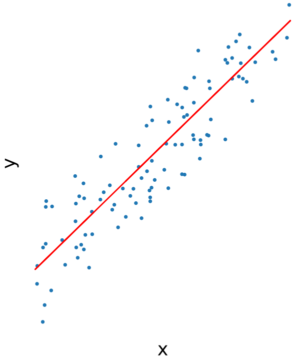
We need to choose an optimizer



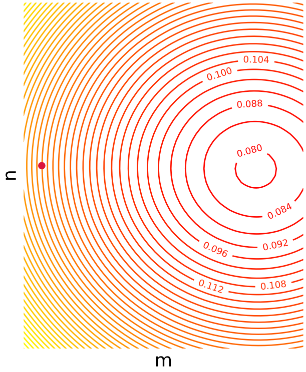


We need to choose an optimizer



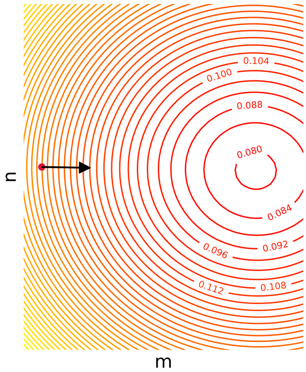


<Stochastic> Gradient Descent



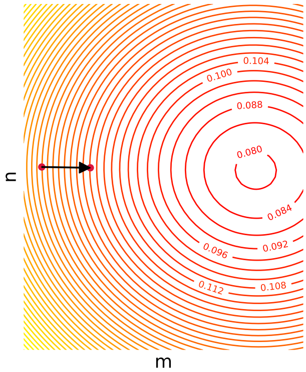
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)

<Stochastic> Gradient Descent



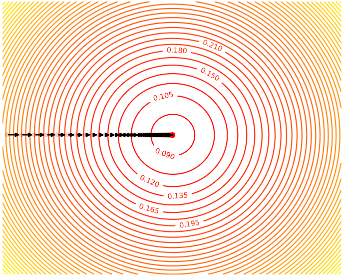
- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)
- Compute the gradients of the loss function with respect to the parameters of the model $\frac{\partial L}{\partial W}\Big|_{\{x,y\}}$ (backpropagation)

<Stochastic> Gradient Descent



- Evaluate the loss function $L = \frac{1}{N}\sum_{i}^{N} l(\hat{y}_i, y_i)$ for a batch of N samples $\{x, y\}$ (forward pass)
- Compute the gradients of the loss function with respect to the parameters of the model $\frac{\partial L}{\partial W}\Big|_{\{x,y\}}$ (backpropagation)
- Update the parameters $W_t = W_{t-1} - \eta \frac{\partial L}{\partial W}\Big|_{\{x,y\}_{t-1}}$

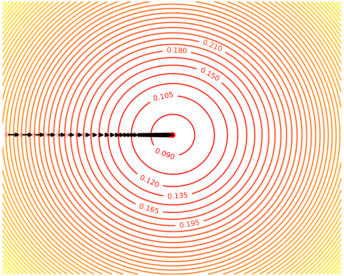
<Stochastic> Gradient Descent



Gradient Descent

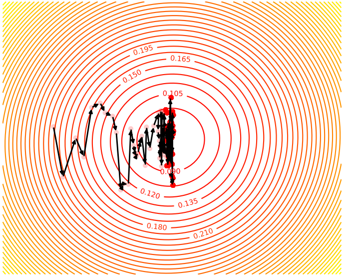
`batch_size = training_set_size`

<Stochastic> Gradient Descent



Gradient
Descent

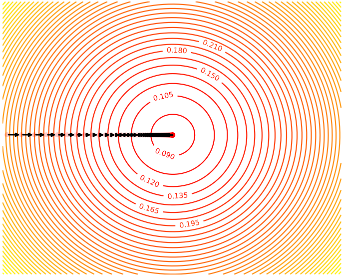
`batch_size = training_set_size`



Stochastic Gradient
Descent

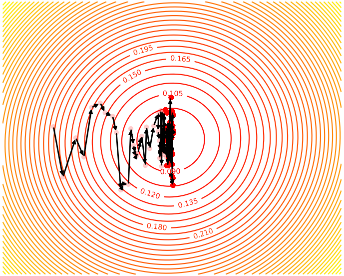
`batch_size = 1`

<Stochastic> Gradient Descent



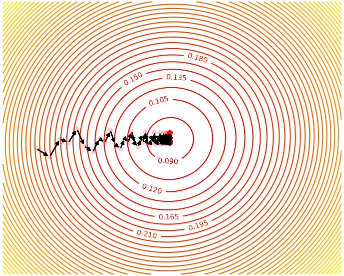
Gradient Descent

`batch_size = training_set_size`



Stochastic Gradient Descent

`batch_size = 1`



Minibatch Stochastic Gradient Descent

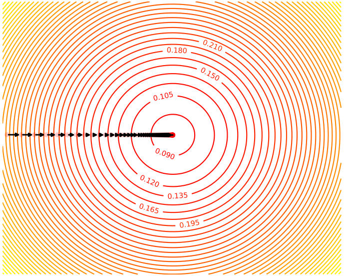
`1 < batch_size < training_set_size`

[lab] Simple Stochastic Gradient Descent

Let's run the notebook `sgd/1-linear_regression_sgd_single_gpu.ipynb`. There we use an unidimensional linear model to understand the trajectories of the SGD minimization.

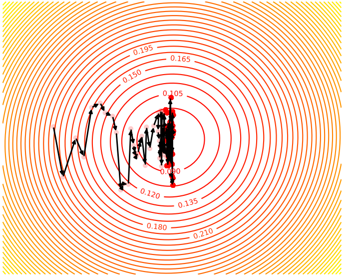
Let's try different batch sizes and see how the trajectory changes.

<Stochastic> Gradient Descent



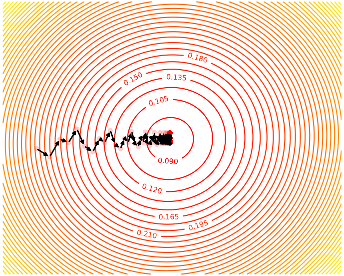
Gradient Descent

`batch_size = training_set_size`



Stochastic Gradient Descent

`batch_size = 1`



Minibatch Stochastic Gradient Descent

`1 < batch_size < training_set_size`

Why training with multiple GPUs?

- The batch size is a hyperparameter

Why training with multiple GPUs?

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory

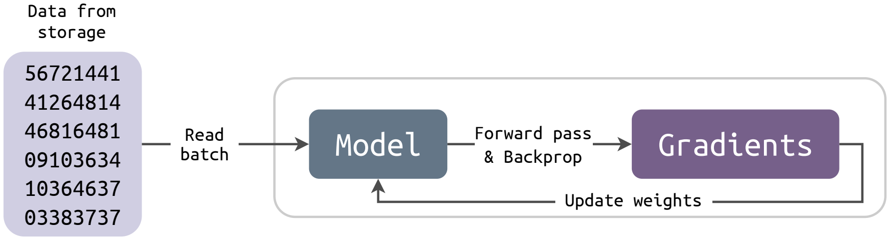
Why training with multiple GPUs?

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple nodes/GPUs enables the use of large batches

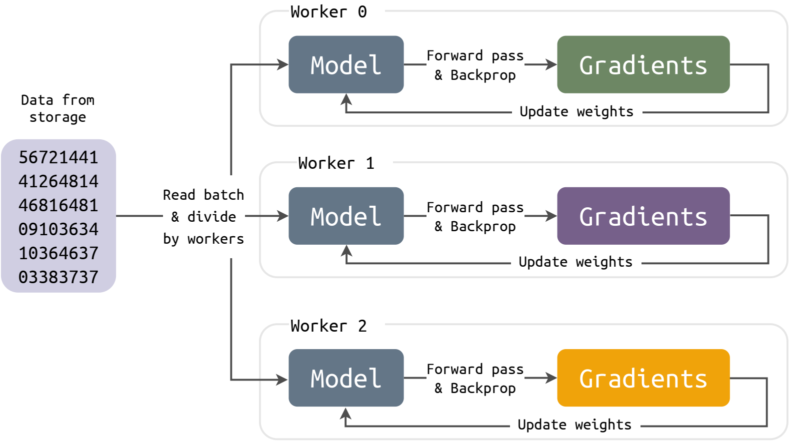
Why training with multiple GPUs?

- The batch size is a hyperparameter
- Large batches may not fit on the GPU memory
- Splitting the training into multiple nodes/GPUs enables the use of large batches
- Multiple nodes/GPUs does not necessarily mean more throughput or faster convergence!

Distributing the training with data parallelism

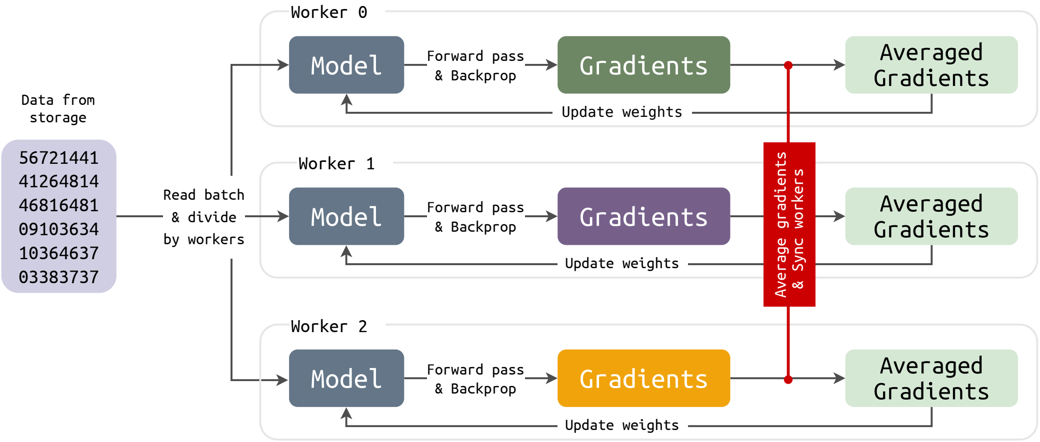


Distributing the training with data parallelism



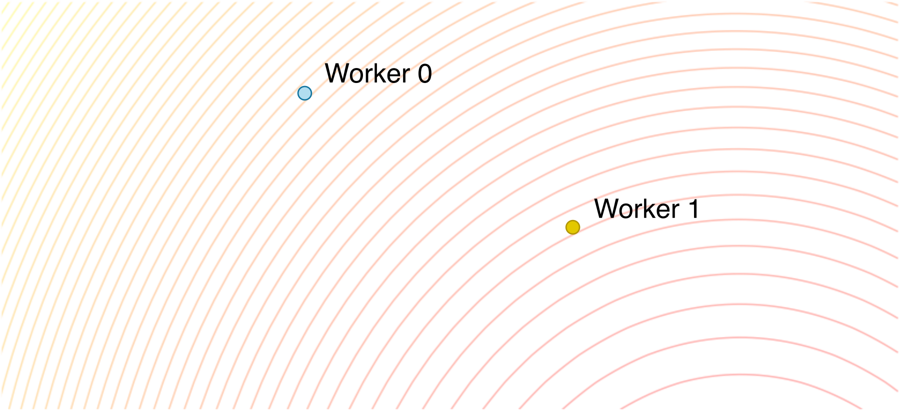
A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Distributing the training with data parallelism



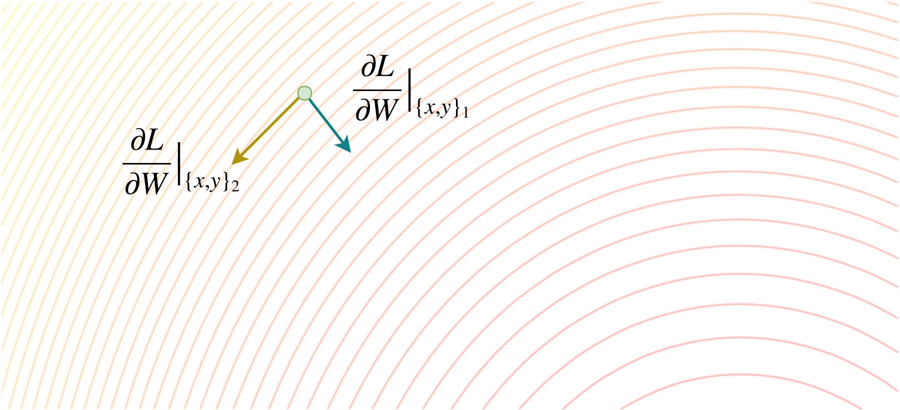
A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

The optimization path with distributed SGD

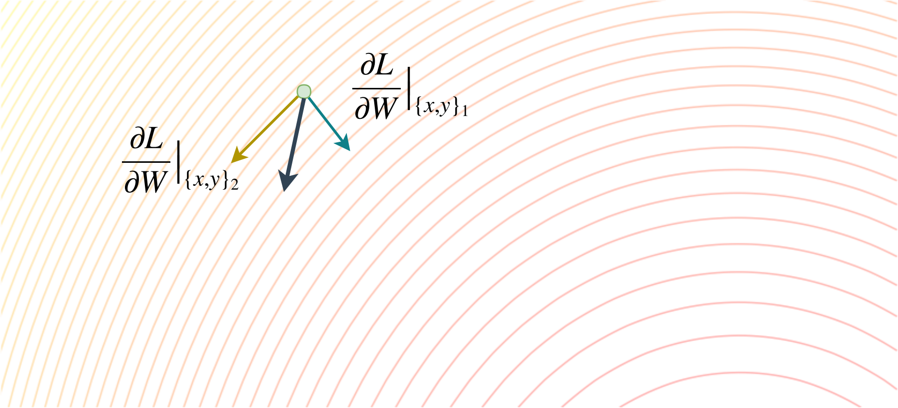


The optimization path with distributed SGD

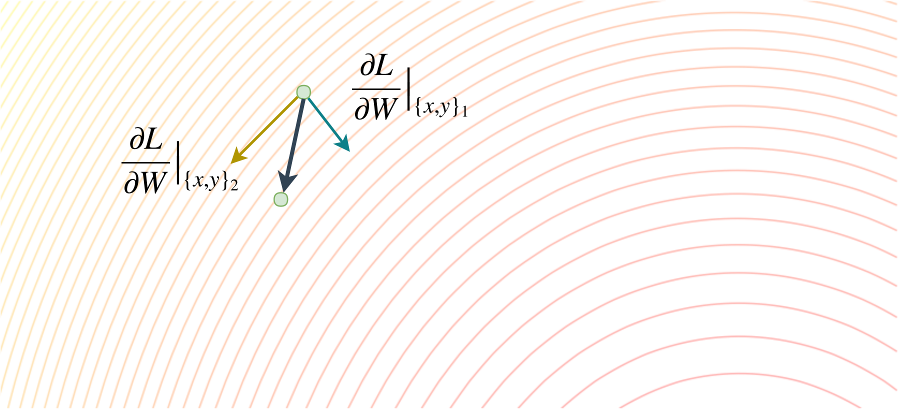
The optimization path with distributed SGD



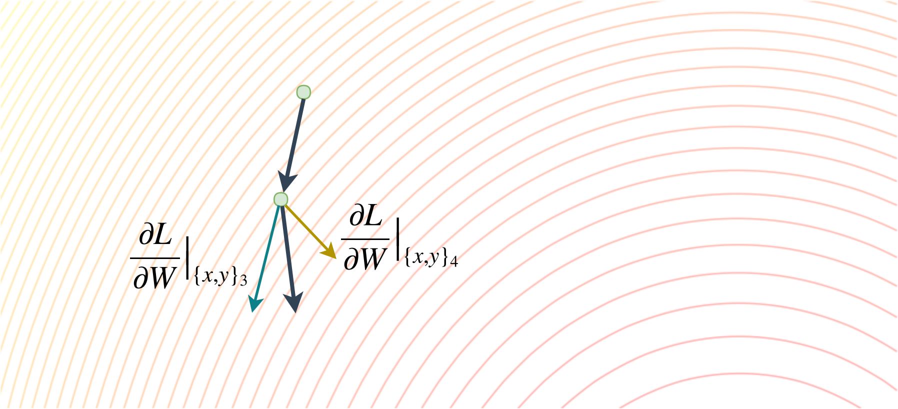
The optimization path with distributed SGD



The optimization path with distributed SGD



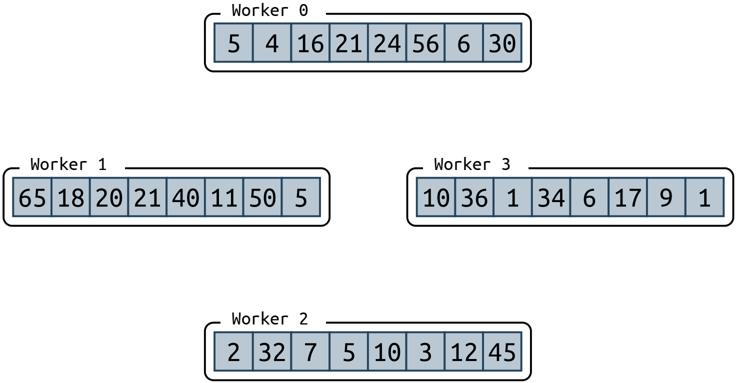
The optimization path with distributed SGD



The Allreduce operation

- The Allreduce name comes from the MPI standard.
- MPI defines the function `MPI_Allreduce` to reduce values from all ranks and broadcast the result of the reduction such that all processes have a copy of it at the end of the operation.
- Allreduce can be implemented in different ways depending on the problem.

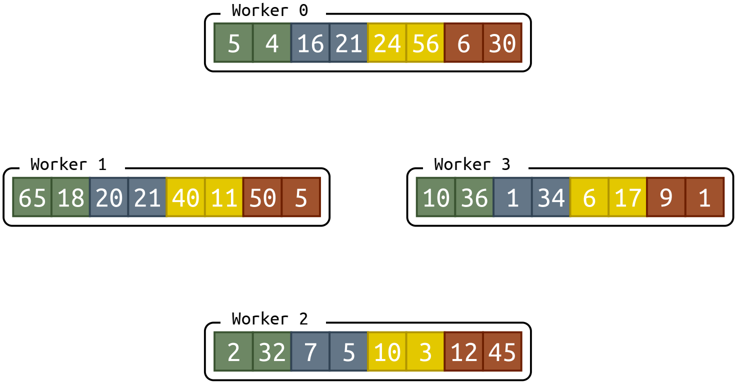
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

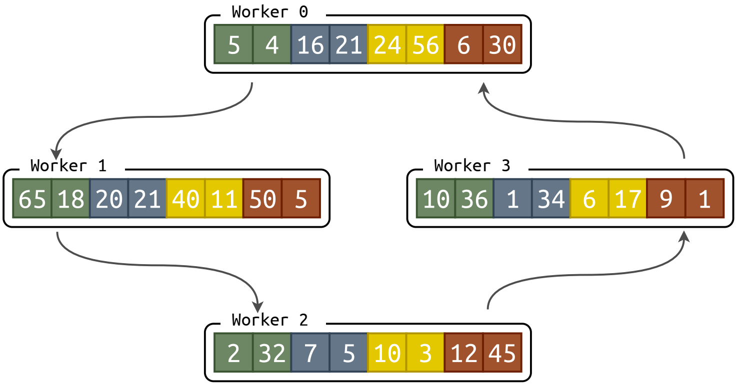
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

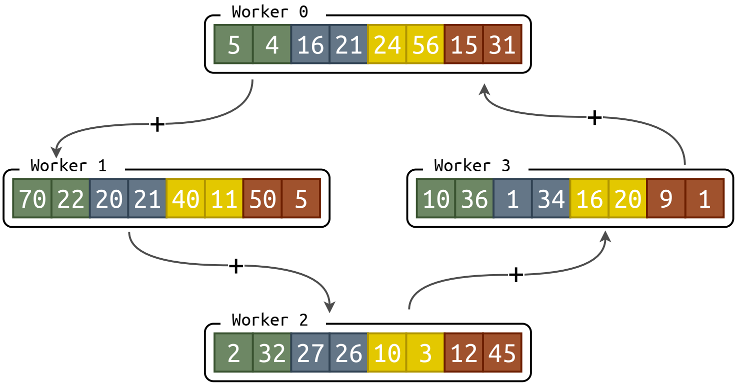
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

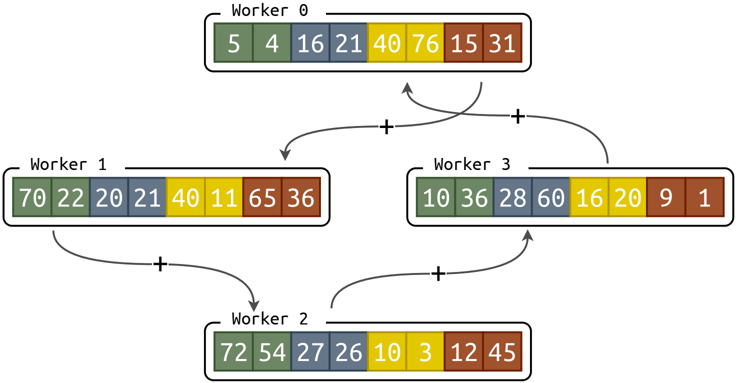
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

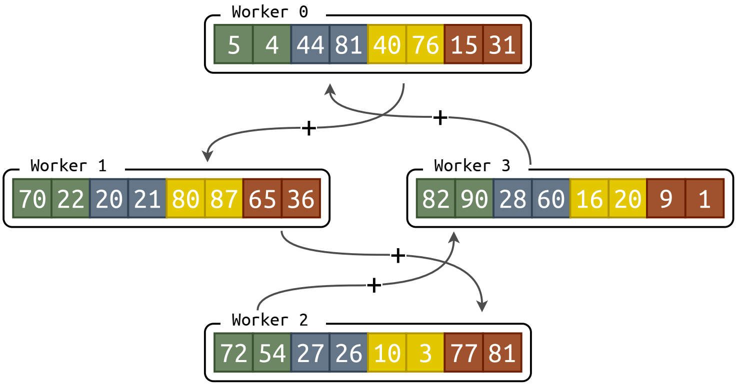
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

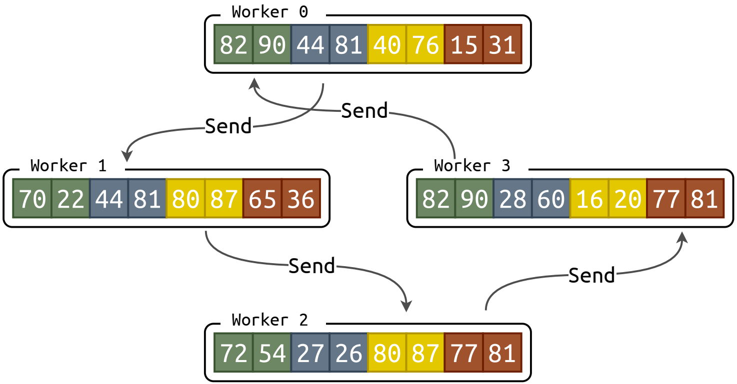
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

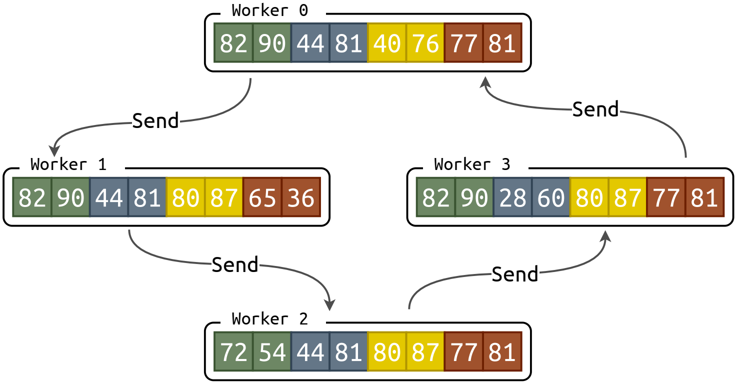
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

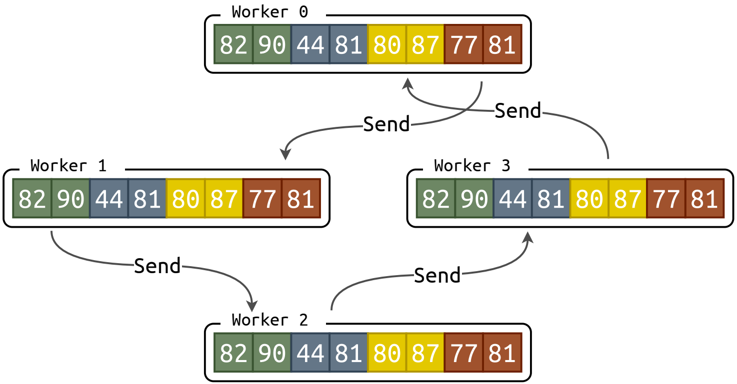
Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Ring Allreduce



Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Ring Allreduce

- Each of the N workers communicates only with two other workers $2(N-1)$ times.
- The values of the reduction are obtained with the first $N-1$ communications.
- The second $N-1$ communications are performed to update the reduced values on all workers.
- The total amount of data sent by each worker $\left[2(N-1)\frac{\texttt{array\_size}}{N}\right]$ is virtually independent of the number of workers.

Baidu-Allreduce on GitHub

A. Sergeev, M. del Balse. Horovod: fast and easy distributed deep learning in TensorFlow

Communication between Cray XC50 Nodes on Piz Daint

- Aries interconnect with the Dragonfly topology
- Direct communication between nodes on the same electrical group (2 cabinets / 384 nodes)
- Communication between nodes on different electrical groups passes by switches (submit with the option `#SBATCH --switches=1` to make your job wait for a single-group allocation)
- More info at CSCS user portal

NVIDIA Collective Communications Library (NCCL)

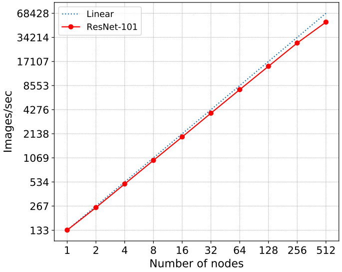
- NCCL implements multi-GPU and multi-node collective communication primitives that are performance optimized for NVIDIA GPUs
- It provides routines such as Allgather, Allreduce and Broadcast, optimized to achieve high bandwidth over PCIe and NVLink high-speed interconnect

Horovod is an open-source distributed training framework for TensorFlow, Keras, PyTorch, and MXNet developed by Uber. The goal of Horovod is to make distributed Deep Learning fast and easy to use

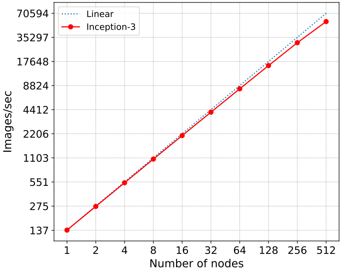
- Minimal code modification required
- Uses bandwidth-optimal communication protocols
- Seamless integration with Cray-MPICH and use of the NVidia Collective Communications Library (NCCL-2)
- Actively developed
- Growing community

- `torch.nn.parallel.DistributedDataParallel` enables multi-node data parallelism with minimum code changes
- `torch.utils.data.DistributedSampler` can be used to split the batch over multiple processes when using `torch.nn.parallel.DistributedDataParallel`
- `torch.distributed` implements the support for sending tensors across processes
- More info at PyTorch's homepage

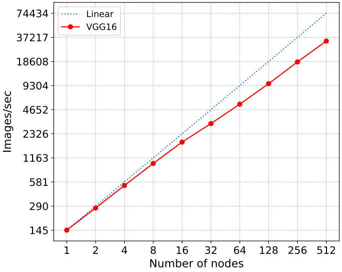
CNNs on Imagenet Benchmark results on Piz Daint (TensorFlow+Horovod)




```
num layers : 347
num weights: 44,601,832
```




```
num layers : 313
num weights: 23,817,352
```




```
num layers : 23
num weights: 138,357,544
```

https://github.com/tensorflow/benchmarks

Running distributed a training on Piz Daint

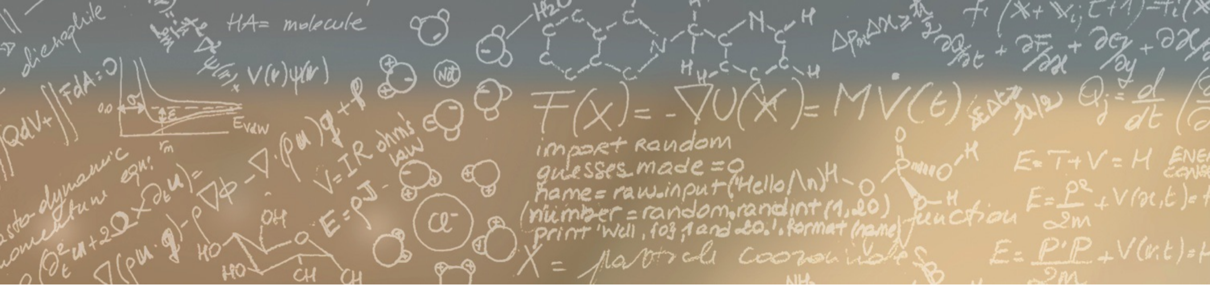
```
#!/bin/bash -l
#SBATCH --job-name=train_distr
#SBATCH --time=00:15:00
#SBATCH --nodes=16
#SBATCH --ntasks-per-core=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --hint=nomultithread
#SBATCH --constraint=gpu
#SBATCH --account=<account>

module load daint-gpu
module load PyTorch # or TensorFlow Horovod
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

export NCCL_DEBUG=INFO
export NCCL_IB_HCA=ipogif0

srun python my_script.py
```

Some additional considerations

- Data must be split equally by workers to avoid load imbalance.
- If applicable, data can be split such that each worker does not need to read all files.
- Consider scaling the learning rate (`lr * dist.get_world_size()`)

[lab] Simple Distributed SGD with TensorFlow and Horovod

We continue with the notebook `sgd/2-exercise-linear_regression_sgd_horovod.ipynb` that uses the same model that we saw in the previous lab. The solution is given in the notebook `sgd/2-solution-linear_regression_sgd_horovod.ipynb`



Thank you for your attention!