intel AI

DISTRIBUTED DL/ML SOLUTIONS FOR HPC SYSTEMS

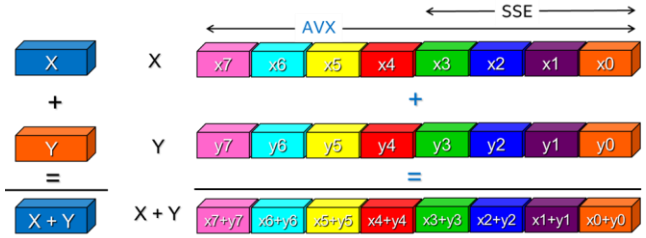
Dr. Fabio Baruffa

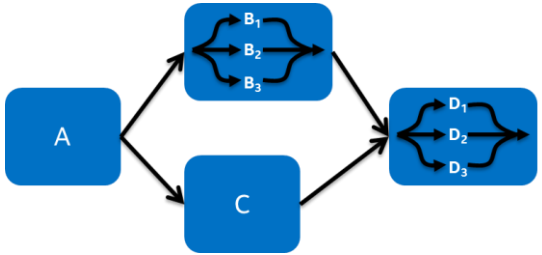
Sr. Technical Consulting Engineer, Intel IAGS

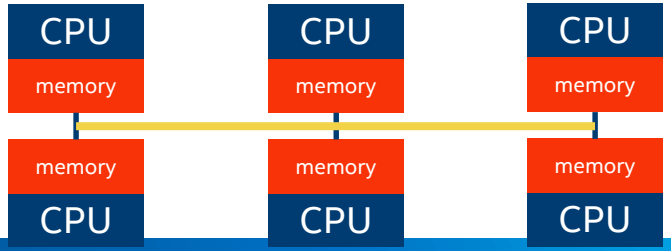
BASIC CONCEPTS ON DISTRIBUTED COMPUTING

TYPES OF PARALLELISM

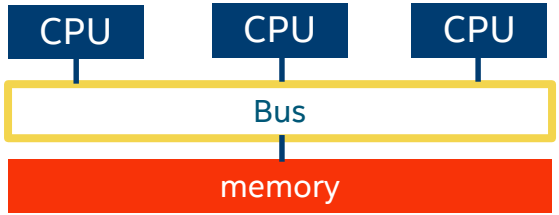
- **SIMD**: Single instruction multiple data (Data Parallel)
  - The same instruction is simultaneously applied on multiple data items
- **MIMD**: Multiple instructions multiple data (Task Parallel)
  - Different instructions on different data
- **SPMD**: Single program multiple data (MPI Parallel)
  - This is the message passing programming on distributed systems



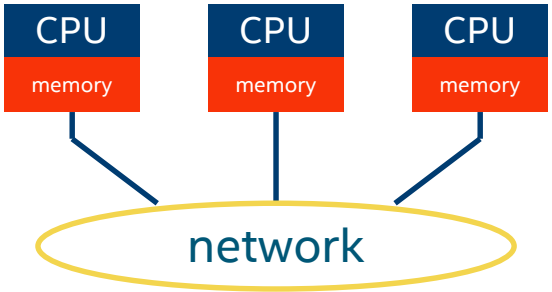




SHARED VS DISTRIBUTED MEMORY SYSTEM



- **Shared memory**
  - There is a unique address space shared between the processors
  - All the processors can access the same memory

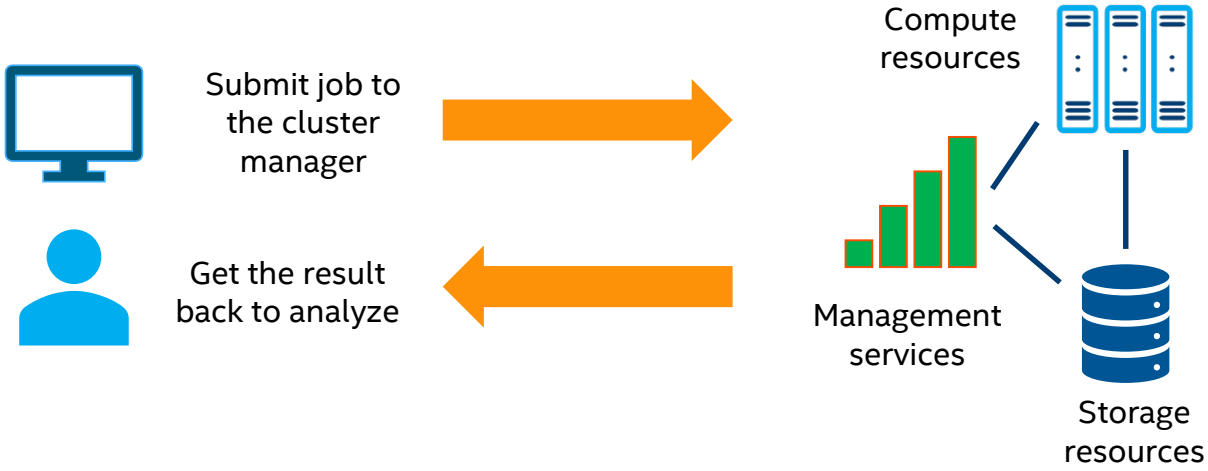


- **Distributed memory**
  - Each processor has its own local memory
  - Messages are exchanged between the processors to communicate the data

WHAT IS HIGH-PERFORMANCE COMPUTING (HPC)?

Leveraging distributed compute resources to solve complex problems with large datasets

- Terabytes to petabytes to zettabytes of data
- Results in minutes to hours instead of days or weeks

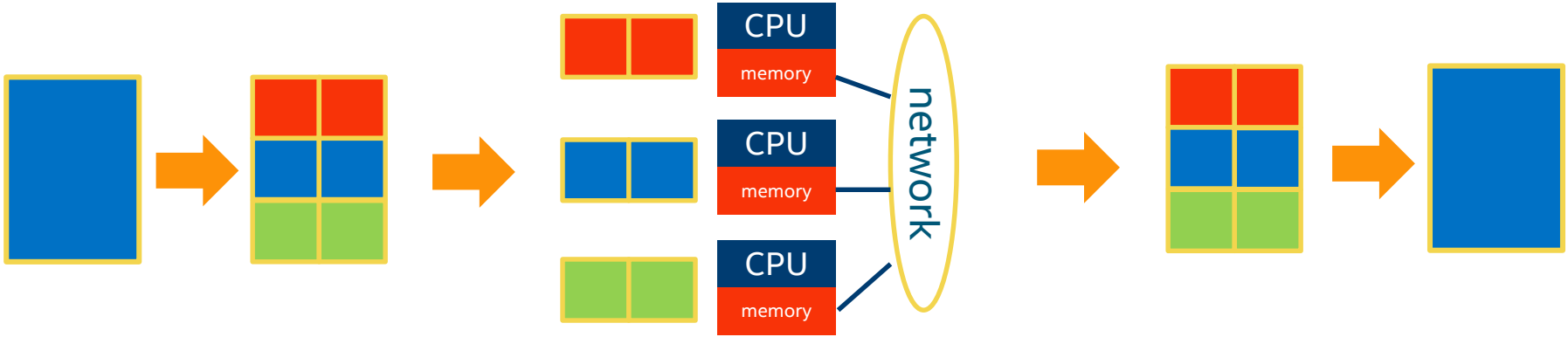


Cluster manager runs workloads on distributed resources, such as CPUs, FPGAs, GPUs and disk drives all interconnected via network

DOMAIN DECOMPOSITION METHOD FOR HPC

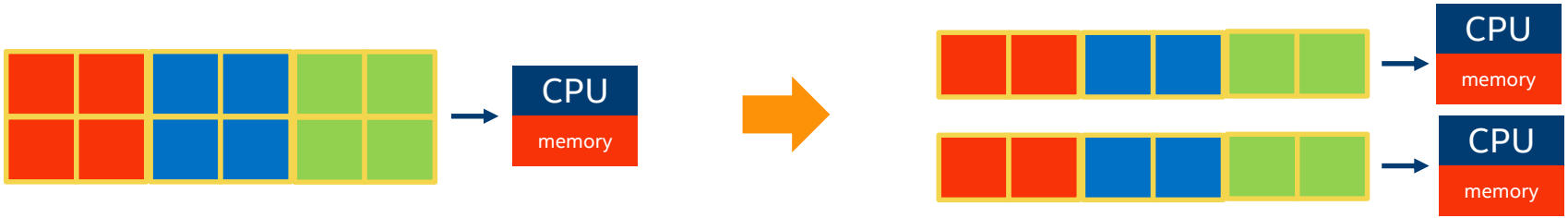
The domain decomposition is a technique for dividing a computational problem in several parts (domains) allowing to solve a large problem on the available resources

- *Partition* the data, assign them to each resource and associate the computation
- *Communication* happens to eventually exchange intermediate results
- *Aggregate* the results from the different resources

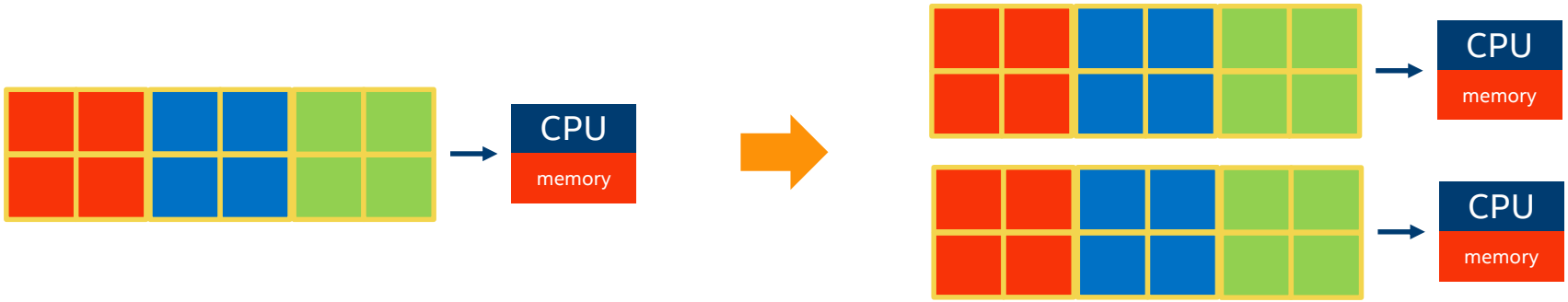


SCALING ASPECTS OF DISTRIBUTED COMPUTING

- Strong scaling: how the time to solution changes by increasing the compute resources for a fixed *total* problem size

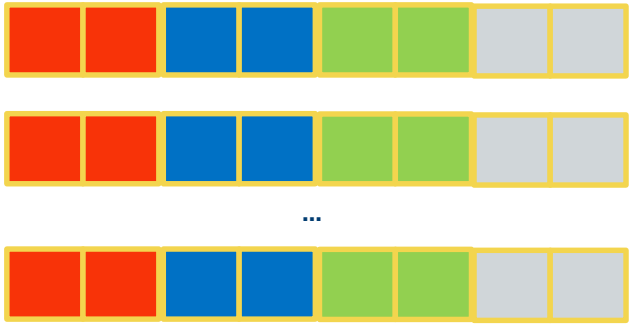


- Weak scaling: how the time to solution changes by increasing the compute resource for a constant problem size *per process*



HOW DO WE REDUCE THE COMPUTATIONAL TIME?

Number of training data set = 8

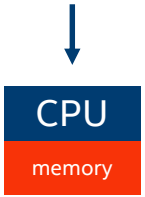


We could use a strong scaling approach to reduce the time for all the epochs

STRONG SCALING ON TRAINING SET

Number of training data set = 8

Epoch 1

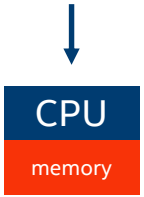


We divide the dataset of 8 training samples into 2 batches of Batch size 4

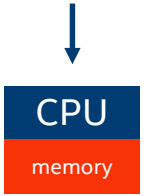
Model will be updated after each batch of 4 samples

Batch size = 4

Batch #1



Batch #2



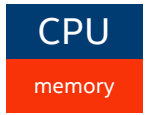
Going beyond reducing the Batch size and increasing the #CPUS (strong scaling) can cause a loose of performance in the model

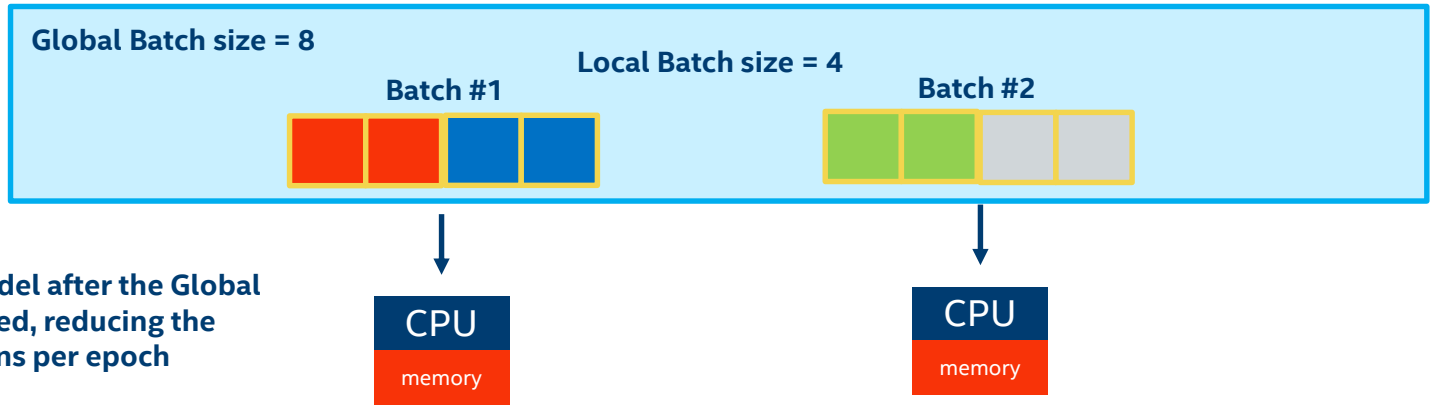
STRONG AND WEAK SCALING ON TRAINING SET

Number of training data set = 8

Epoch 1

We keep the same Batch size/CPU, increasing the overall Batch size





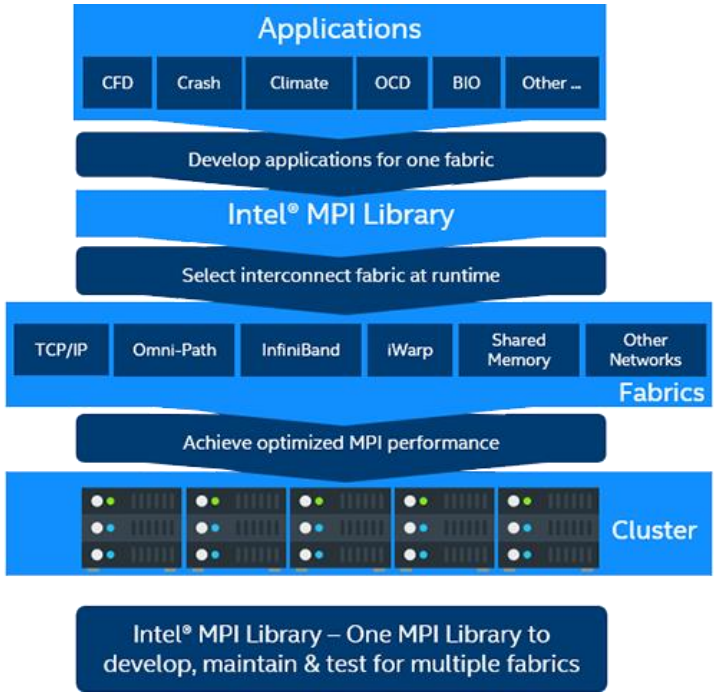
We update the model after the Global batch size is reached, reducing the number of iterations per epoch

MESSAGE PASSING INTERFACE (MPI)

MPI is a standard which gets implemented in form of libraries for inter-process communication and data exchange.

Function categories:

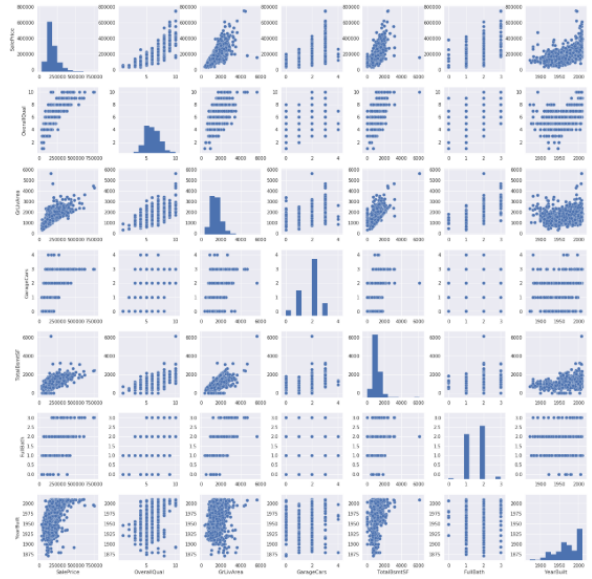
- Point-to-point communication
- Collective communication
- Communicator topologies
- User-defined data types
- Utilities (for example, timing and initialization)



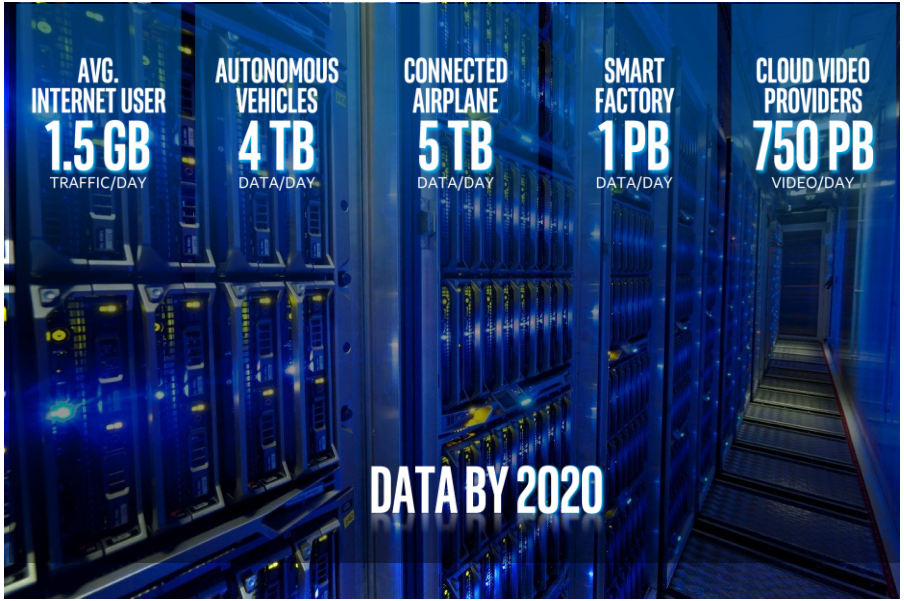
DISTRIBUTING STRATEGY FOR MACHINE LEARNING

FROM PROTOTYPE TO PRODUCTION

In [13]:
```
#scatterplot
sns.set()
cols = ['SalePrice', 'OverallQual', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath', 'YearBuilt']
sns.pairplot(df_train[cols], size = 2.5)
plt.show();
```



PERFORMANCE



https://www.kaggle.com/pmarcelino/comprehensive-data-exploration-with-python

WHY DISTRIBUTED ML/DL

- Most Machine Learning tasks assume the data can be easily accessible, but:
  - Data loading on a single machine can be a bottleneck in case of large amount of data
  - To run production applications large memory systems is required (data not fitting in the local computer RAM)
  - Traditional sequential algorithms are not suitable in case of distributed memory system
- Time to solution is critical on highly competitive market.

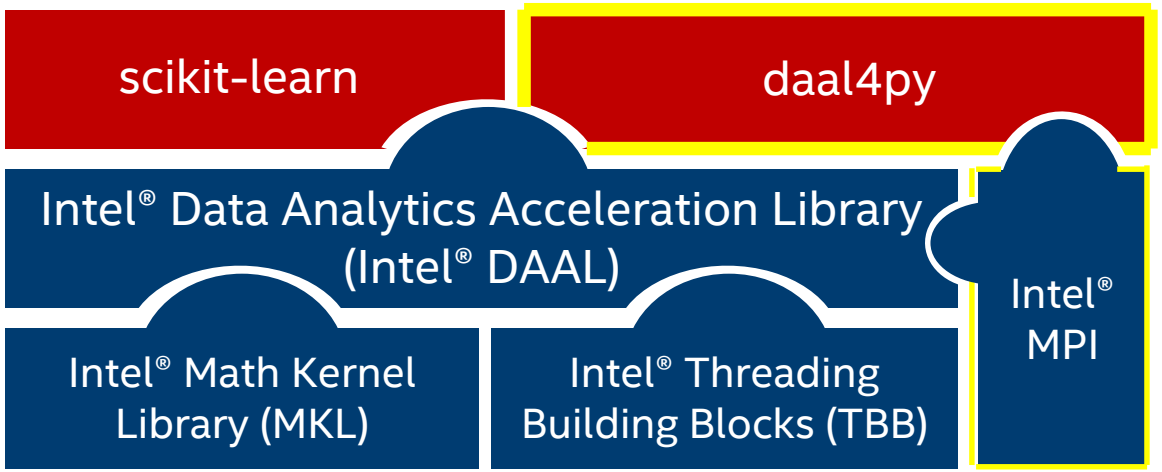
WHY DISTRIBUTED ML/DL

- Deep Learning training takes time:
  - Computational complexity of DL training can be up to 100+ ExaFLOP (1 ExaFLOP =10^{18} op);
  - Typical single node performance is up-to tens of TeraFLOPS (1 TF = 10^{12} op/sec);
  - Peak performance of most powerful HPC clusters is up-to tens of PetaFLOPS (1 PF = 10^{15} op/sec).
- **Time to solution is critical on highly competitive market.**

DAAL4PY: ACCELERATED ANALYTICS TOOLS

- Package created to address the needs of **Data Scientists and Framework Designers** to harness the **Intel® Data Analytics Acceleration Library (DAAL)** with a **Pythonic API**
- For scaling capabilities, **daal4py** also provides the ability to do distributed machine learning using **Intel® MPI library**
- **daal4py** operates in SPMD style (Single Program Multiple Data), which means your program is executed on several processes (e.g. similar to MPI)
- The use of MPI is not required for **daal4py**'s SPMD-mode to work, all necessary communication and synchronization happens under the hood of daal4py
- It is possible to use **daal4py** and mpi4py in the same program

SCALING MACHINE LEARNING BEYOND A SINGLE NODE



Simple Python API
Powers scikit-learn

Powered by DAAL

Scalable to multiple nodes

Try it out! `conda install -c intel daal4py`

K-MEANS USING DAAL4PY

```
import daal4py as d4p

# daal4py accepts data as CSV files, numpy arrays or pandas dataframes
# here we let daal4py load process-local data from csv files
data = "kmeans_dense.csv"

# Create algob object to compute initial centers
init = d4p.kmeans_init(10, method="plusPlusDense")
# compute initial centers
ires = init.compute(data)
# results can have multiple attributes, we need centroids
Centroids = ires.centroids
# compute initial centroids & kmeans clustering
result = d4p.kmeans(10).compute(data, centroids)
```

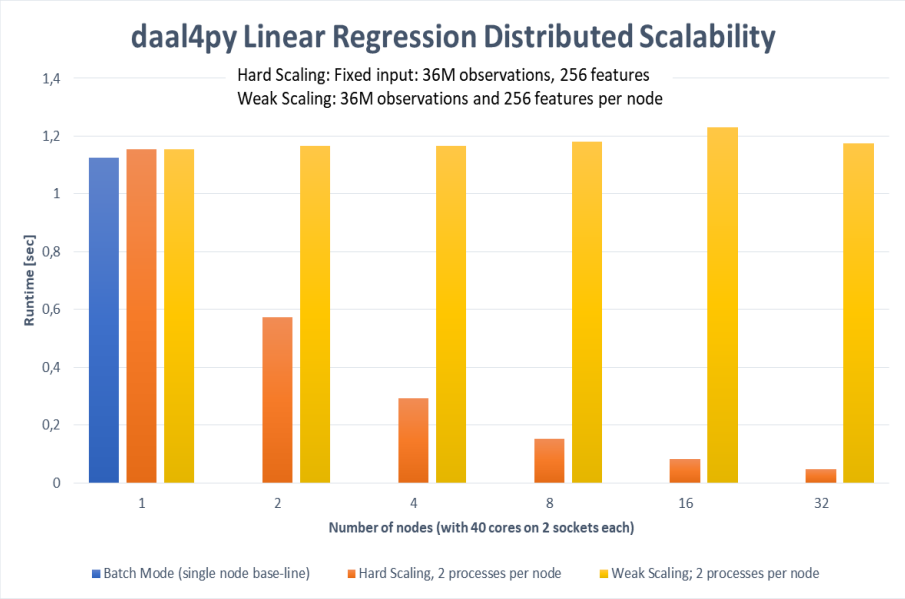
DISTRIBUTED K-MEANS USING DAAL4PY

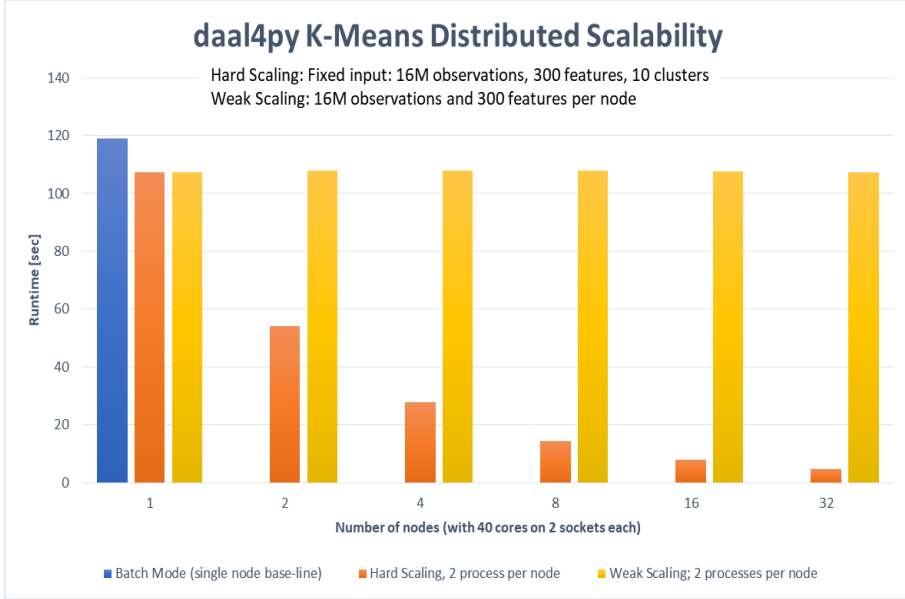
```
import daal4py as d4p

# initialize distributed execution environment
d4p.daalinit()

# daal4py accepts data as CSV files, numpy arrays or pandas dataframes
# here we let daal4py load process-local data from csv files
data = "kmeans_dense_{}.csv".format(d4p.my_procid())

# compute initial centroids & kmeans clustering
init = d4p.kmeans_init(10, method="plusPlusDense", distributed=True)
centroids = init.compute(data).centroids
result = d4p.kmeans(10, distributed=True).compute(data, centroids)
```

```
mpirun -n 4 python ./kmeans.py
```

STRONG & WEAK SCALING VIA DAAL4PY

Hardware	Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, EIST/Turbo on
	2 sockets, 20 Cores per socket
	192 GB RAM
	16 nodes connected with Infiniband
Operating System	Oracle Linux Server release 7.4
Data Type	double





On a 32-node cluster (1280 cores) daal4py computed linear regression of 2.15 TB of data in 1.18 seconds and 68.66 GB of data in less than 48 milliseconds.

On a 32-node cluster (1280 cores) daal4py computed K-Means (10 clusters) of 1.12 TB of data in 107.4 seconds and 35.76 GB of data in 4.8 seconds.

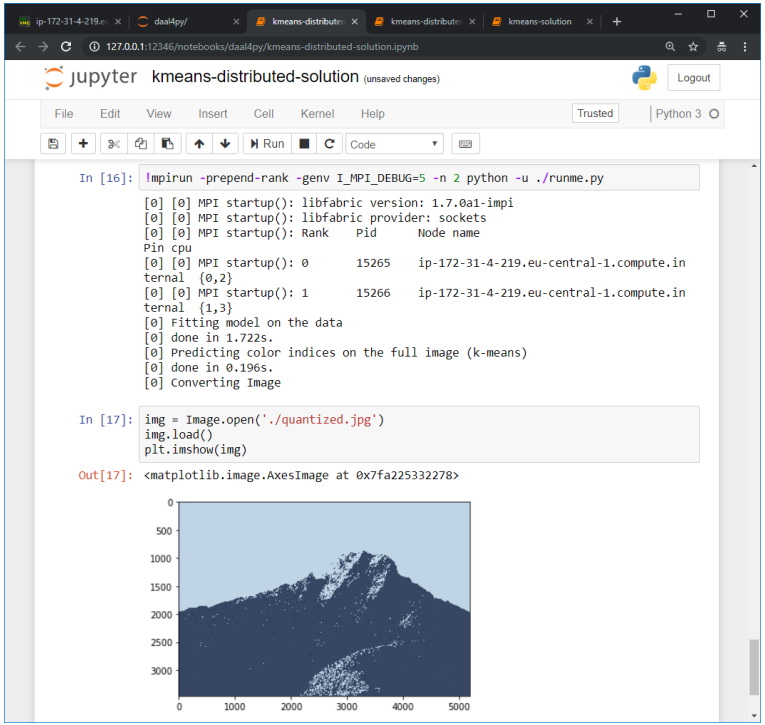
HANDS-ON

DISTRIBUTED K-MEANS USING DAAL4PY

1) Performs a pixel-wise Vector Quantization (VQ) using K-Means
2) Implemented the domain decomposition according to:
   - `d4p.num_procs()`
   - `d4p.my_procid()`
3) Using the distributed algorithm from Daal4Py
   - `d4p.kmeans_init(n_colors, method="plusPlusDense", distributed=True)`
4) What is the meaning of `d4p.daalinit()` & `d4p.daalfini()`?
5) How does threading compare to multiprocessing in terms of performance?

DISTRIBUTED K-MEANS SUMMARY

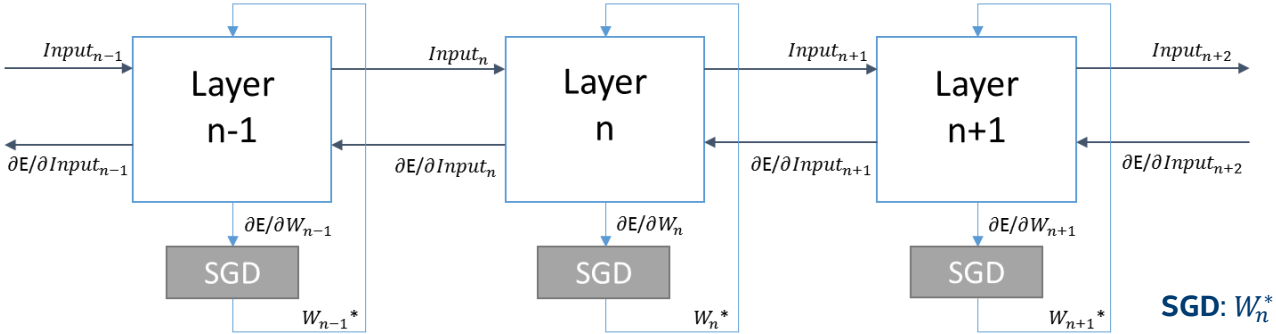
- Each process (MPI rank) get's a different chunk of data
- Only process #0 reports results
- Inference is using the same routines as training with 0 maximum iterations and centroid assignment
- There is no oversubscription since DAAL only sees the cores "owned" by the corresponding MPI rank



DISTRIBUTING STRATEGY FOR DEEP LEARNING

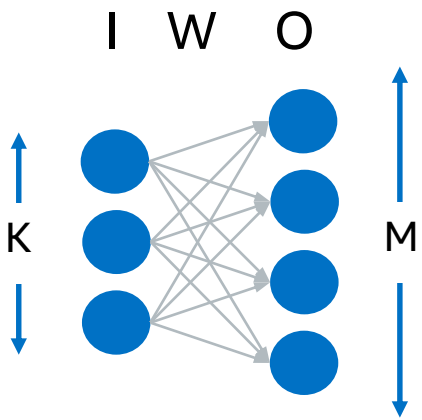
DEEP LEARNING TRAINING PROCEDURE

- **Forward propagation**: calculate loss function based on the input batch and current weights;
- **Backward propagation**: calculate error gradients w.r.t. weights for all layers (using chain rule);
- **Weights update**: use gradients to update weights; there are different algorithms exist - vanilla SGD, Momentum, Adam, etc.



SGD: $W_n^* = W_n - \alpha * \partial E/\partial W_n$ or variants

PARALLELISM OPTIONS

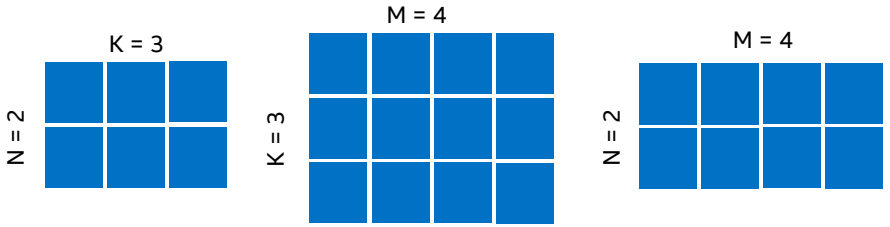


Fully connected layer

$I \in R^{NxK}$
Input

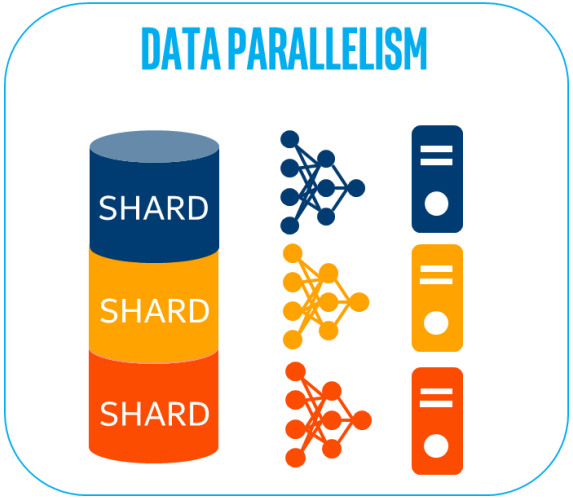
$W \in R^{KxM}$
Weights or model

$O \in R^{NxM}$
Output or activations

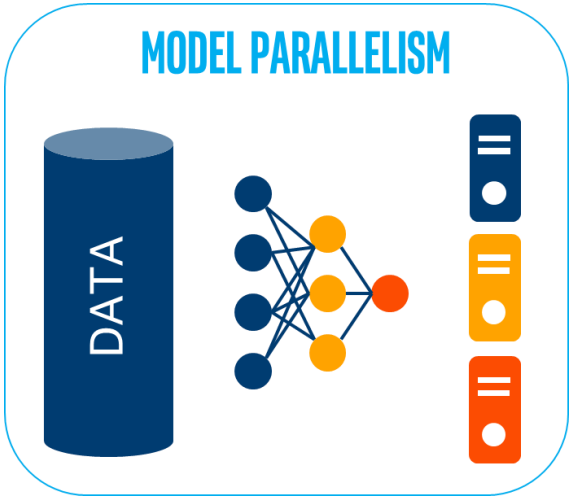


Several options for parallelization

NEURAL NETWORK PARALLELISM



Data is processed in increments of N. Work on minibatch samples and distributed among the available resources.

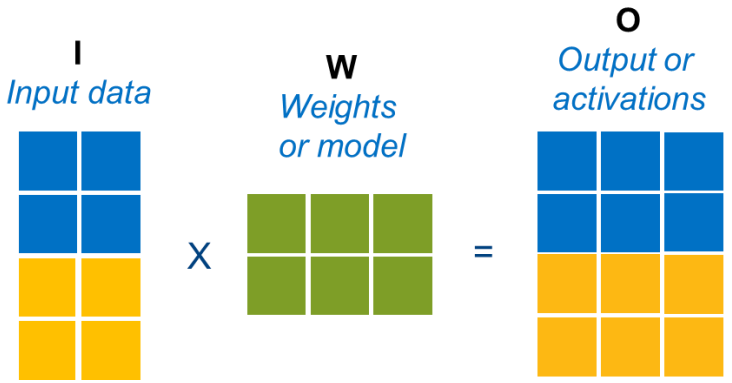


The work is divided according to the neurons in each layer. The sample minibatch is copied to all processors which compute part of the DNN.

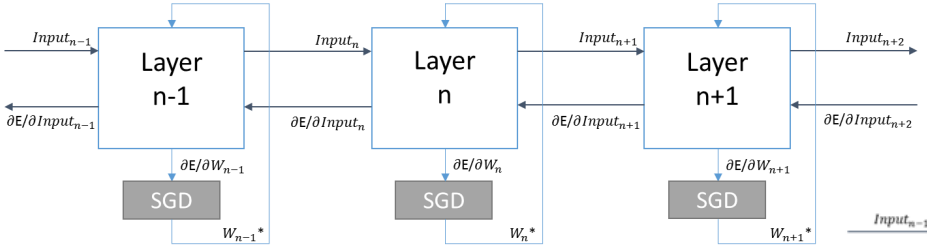
source: https://arxiv.org/pdf/1802.09941.pdf

MULTI-NODE PARALLELIZATION

- **Data parallelism**:
  - Replicate the model across nodes;
  - Feed each node with its own batch of input data;
  - Communication for gradients is required to get their average across nodes;
  - Can be either
    - *AllReduce* pattern
    - *ReduceScatter* + *AllGather* patterns



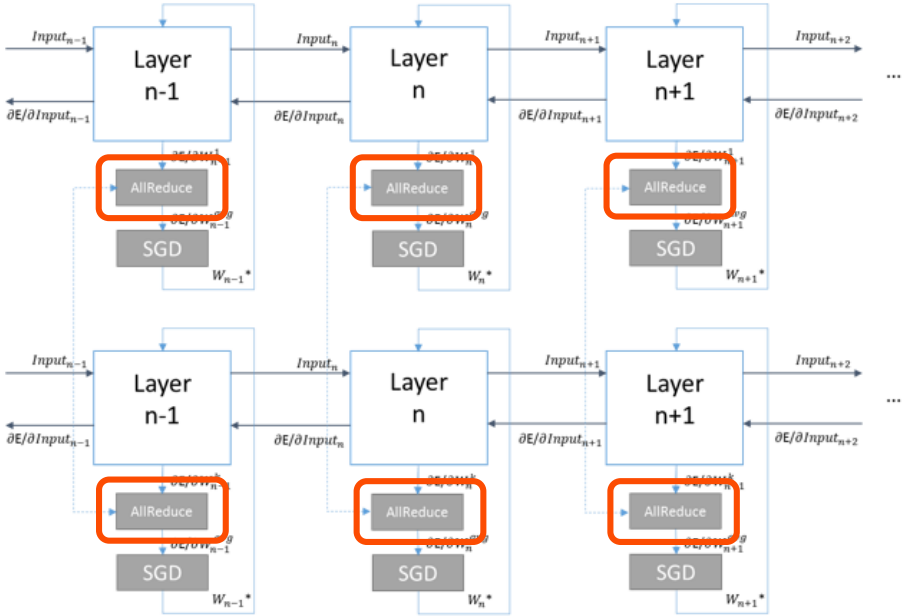
DATA PARALLELISM



... $E(I, W)$

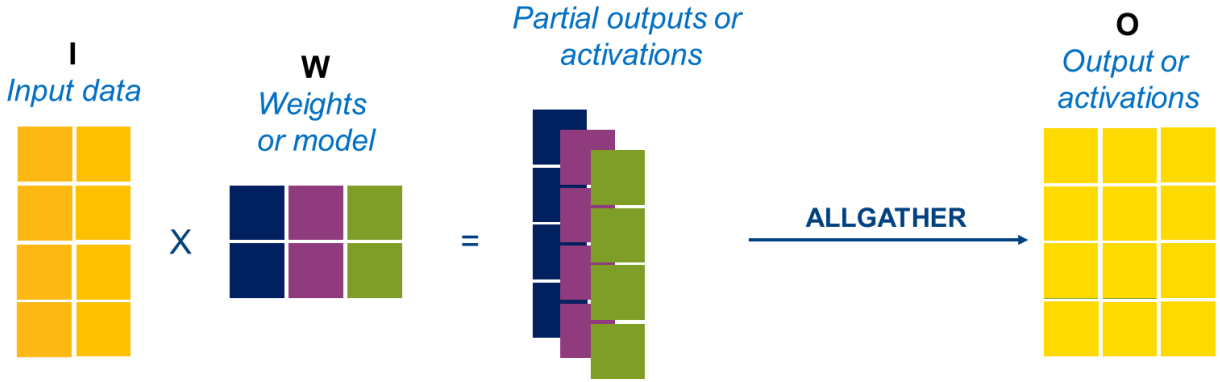
Single node

Multi node
Data Parallelism



MULTI-NODE PARALLELIZATION

- **Model parallelism**:
  - Model is split across nodes;
  - Feed each node with the same batch of input data;
  - Communication for partial activations is required to gather the result;



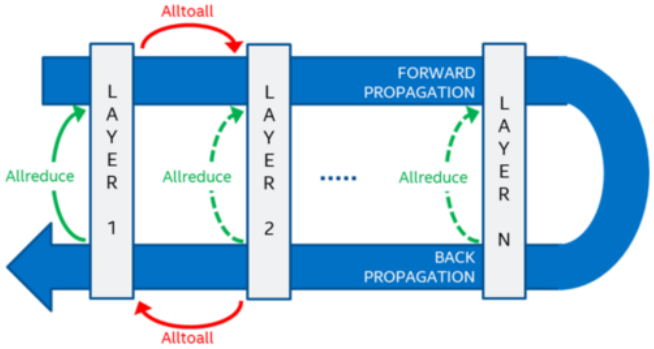
MULTI-NODE PARALLELIZATION

- **What parallelism flavor to use?**
  - Use model parallelism when volume of gradients is much higher than volume of activations or when model doesn't fit memory;
  - Use data parallelism otherwise;
  - Parallelism choice affects activations/gradients ratio
    - Data parallelism at scale makes activations << weights
    - Model parallelism at scale makes weights << activations
  - There're also other parallelism flavors – pipelined, spatial, etc.

INTEL® MACHINE LEARNING SCALING LIBRARY (MLSL)

Distributed Deep Leaning Requirements:

- ✓Compute/communication overlap
- ✓Choosing optimal communication algorithm
- ✓Prioritizing latency-bound communication
- ✓Portable / efficient implementation
- ✓Ease of integration with quantization algorithms
- ✓Integration with Deep Learning Frameworks



Communication dependent on work partitioning strategy
Data parallelism = Allreduce (or) Reduce_Scatter + Allgather
Model parallelism = AlltoAll

Numerous DL Frameworks

Multiple NW Fabrics

Ethernet
OmniPath® Infiniband®

INTEL® MACHINE LEARNING SCALING LIBRARY (MLSL)

https://github.com/01org/MLSL/releases

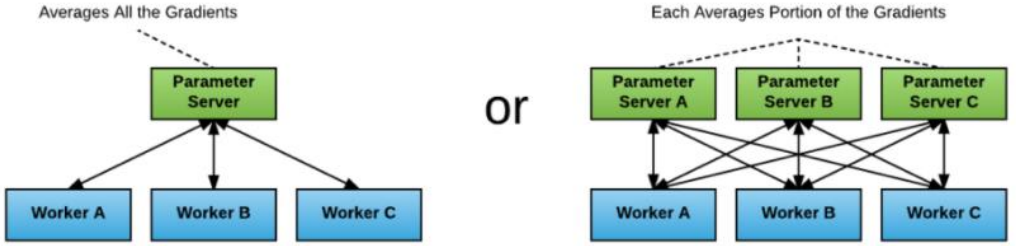
Some of the Intel MLSL features include:

- Built on top of MPI, transparently supports various interconnects: Intel® Omni-Path Architecture, InfiniBand*, and Ethernet;
- Optimized to drive scalability of DL communication patterns
- Ability to trade off compute for communication performance – beneficial for communication-bound scenarios
- New domain-specific features are coming soon

DISTRIBUTED TENSORFLOW*

Distributed Tensorflow with Parameter Server

With Parameter Server

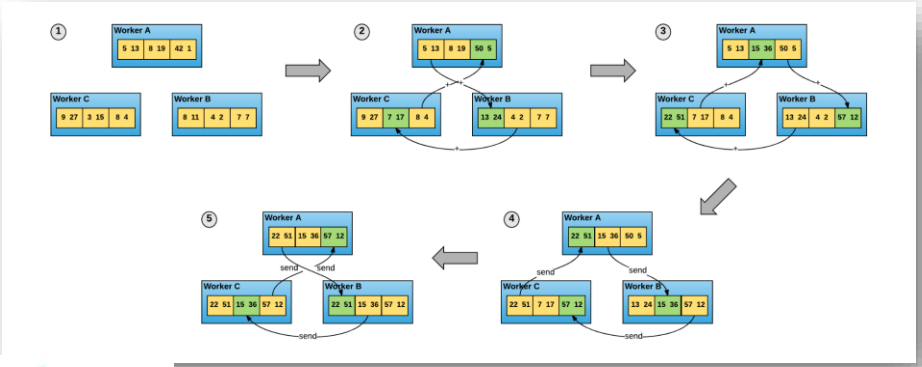


The parameter server model for distributed training jobs can be configured with different ratios of parameter servers to workers, each with different performance profiles.

Source: https://eng.uber.com/horovod/

DISTRIBUTED TENSORFLOW* WITH HOROVOD

ring-allreduce



The ring all-reduce algorithm allows worker nodes to average gradients and disperse them to all nodes without the need for a parameter server.

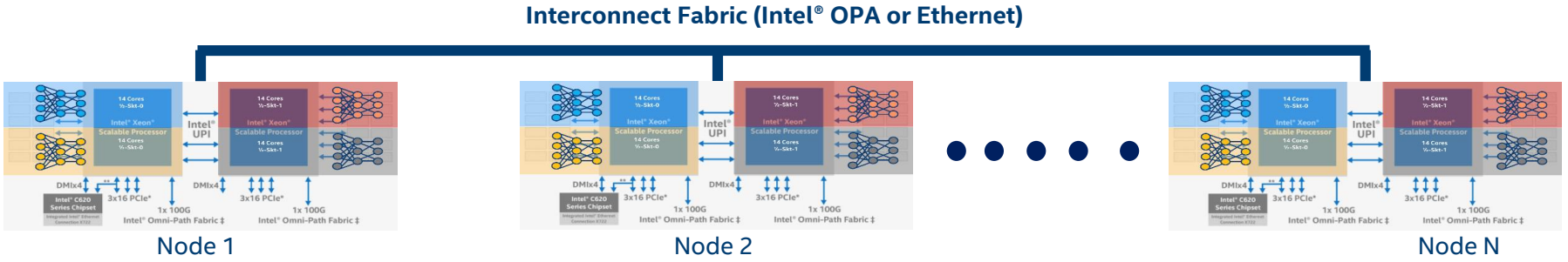
Source: https://eng.uber.com/horovod/

Uber's open source Distributed training framework for TensorFlow

- Horovod is a distributed training framework for TensorFlow, Keras, PyTorch, and MXNet.
- The goal of Horovod is to make distributed Deep Learning fast and easy to use
- Horovod core principles are based on MPI concepts such as **size, rank, local rank, allreduce, allgather and broadcast**.
- Separate infrastructure with model development
- Advantages
  - Minimal code changes to run distributed TensorFlow
  - Network-optimal
  - No parameter server

More info: https://github.com/horovod/horovod/

DISTRIBUTED TRAINING WITH HOROVOD* MPI LIB



Distributed Deep Learning Training Across Multiple nodes

Each node running multiple workers/node

Uses optimized MPI Library for gradient updates over network fabric

Caffe – Use Optimized Intel® MPI ML Scaling Library (Intel® MLSL)

TensorFlow* – Uber horovod MPI Library

Intel Best Known Methods: https://ai.intel.com/accelerating-deep-learning-training-inference-system-level-optimizations/

HOROVOD: HOW TO CHANGE THE CODE

- Add `import horovod.tensorflow as hvd` and run `hvd.init()` in the beginning of the program
- Scale the learning rate by number of workers. Effective batch size in synchronous distributed training is scaled by the number of workers. An increase in learning rate compensates for the increased batch size.
- Wrap optimizer in `hvd.DistributedOptimizer`. The distributed optimizer delegates gradient computation to the original optimizer, averages gradients using **allreduce** or **allgather**, and then applies those averaged gradients.
- Add `hvd.BroadcastGlobalVariablesHook(0)` to broadcast initial variable states from rank 0 to all other processes. This is necessary to ensure consistent initialization of all workers when training is started with random weights or restored from a checkpoint. Alternatively, if you're not using MonitoredTrainingSession, you can simply execute the `hvd.broadcast_global_variables` op after global variables have been initialized.
- Modify your code to save checkpoints only on worker 0 to prevent other workers from corrupting them. This can be accomplished by passing `checkpoint_dir=None` to `tf.train.MonitoredTrainingSession`, if `hvd.rank() != 0`.

https://github.com/horovod/horovod#usage

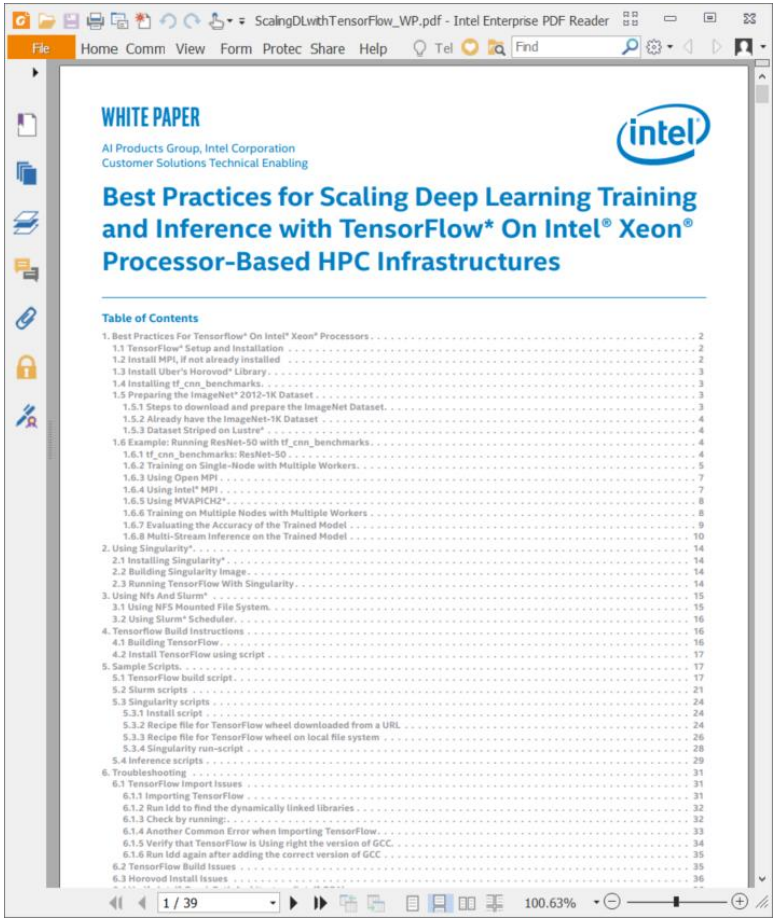
HOROVOD 101 QUICK START


```
import horovod.tensorflow as hvd
hvd.init()

#Scale the optimizer
opt = tf.train.AdagradOptimizer(0.01 * hvd.size())

# Add Horovod Distributed Optimizer
opt = hvd.DistributedOptimizer(opt)

hooks = [hvd.BroadcastGlobalVariablesHook(0)]

# Save checkpoints only on worker 0 to prevent other workers from
corrupting them.
checkpoint_dir = '/tmp/train_logs' if hvd.rank() == 0 else None
```

SCALING TENSORFLOW*

There is way more to consider when striking for peak performance on distributed deep learning training.:

https://ai.intel.com/white-papers/best-known-methods-for-scaling-deep-learning-with-tensorflow-on-intel-xeon-processor-based-clusters/



INTEL® MLSL BACKEND FOR HOROVOD

Install procedure:

- Install the latest versions of Intel MLSL and Intel MPI;
- `source <mlsl_install>/intel64/bin/mlslvars.sh` **`thread`**
- `source <intel_mpi_2019>/intel64/bin/mpivars.sh` **`release_mt`**
- Download Horovod and build it from source code or
  - `pip install horovod`

INTEL® MLSL BACKEND FOR HOROVOD

Launch procedure:

- `export MLSL_LOG_LEVEL=1`
  - output from within MLSL
- `export MLSL_NUM_SERVERS=X`
  - X is the number of cores you'd like to dedicate for driving communication
- `export MLSL_SERVER_AFFINITY=c1,c2,..,cX`
  - Core IDs dedicated to MLSL servers (uses X 'last' cores by default)
- `export HOROVOD_MLSLbackground_BGT_AFFINITY=c0`
  - Affinity for thread of Horovod
- Adjust OpenMP settings to avoid intersection with c0,c1,..,cX

HANDS-ON

TENSORFLOW+HOROVOD/CNN_MNIST-HVD.IPYNB

Delete the checkpoint if needed, otherwise TF won't train any further

- rm -rf checkpoints

Let's start changing the numer of MPI tasks, what performance difference would you expect?

- mpirun -prepend-rank -genv OMP_NUM_THREADS=2 -genv I_MPI_DEBUG=5 -n 2 python -u cnn_mnist-hvd.py
- mpirun -prepend-rank -genv OMP_NUM_THREADS=2 -genv I_MPI_DEBUG=5 -n 4 python -u cnn_mnist-hvd.py
- check the size of the dataset:
  - ls -lha ~/.keras/datasets/

Intel Python and Optimized Tensorflow

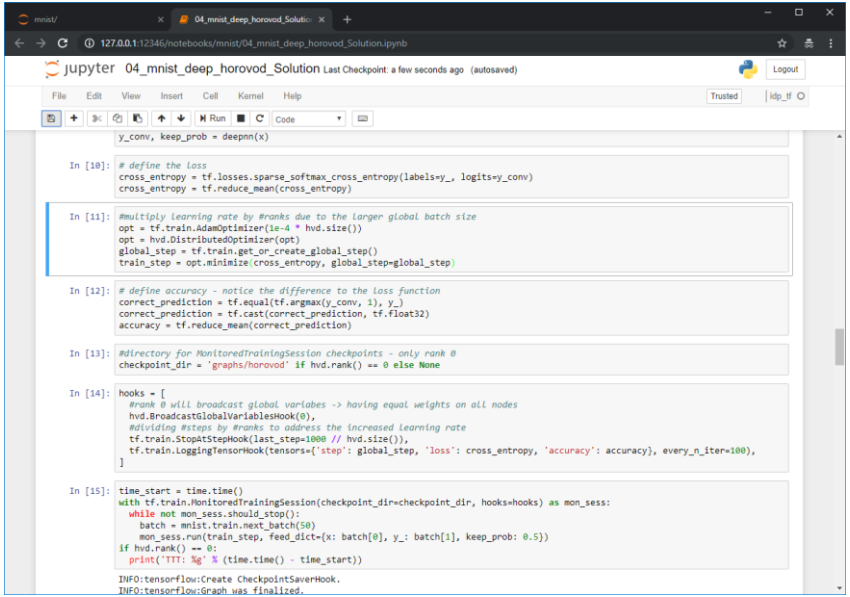
- source activate hvd-impi
- pip show tensorflow | grep Location
  - useful to locate the TF installation for see the library linked: ldd $Location/tensorflow/libtensorflow...so
- rm-rf /tmp/*
- export export MKLDNN_VERBOSE=1

TENSORFLOW+HOROVOD/CNN_MNIST-HVD.IPYNB

1) How to initialize Horovod and why is it necessary?
2) Why is it necessary to adept the learning rate with larger batches?
3) How can you dynamically adept the learning rate?
4) How to identify rank #1 (0)?
5) Why is it necessary to adept the number of training steps according to the number of workers / larger batches?
6) How can you dynamically adept the number of training steps?
7) How is the single process performance vs 2 ranks vs 4 ranks?

MNIST CNN HOROVOD DEMO SUMMARY

- Horovod initializes the MPI communication underneath and therefore defines rank() and size()
- In order to reduce the Time To Train with multiple workers, therefore increasing the batch size, the learning rate needs to scale
- Same for the # of steps for training
- 4 ranks can be faster since less threading efficiency is required in small convolutions



Legal Disclaimer & Optimization Notice

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

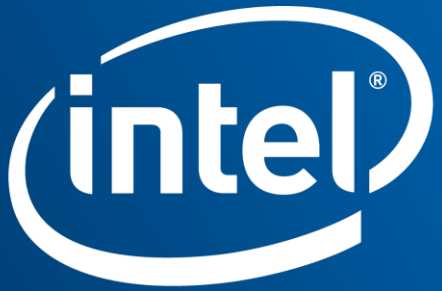
INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

BACKUP



Software