Distributed Deep Learning

Huihuo Zheng

Data Science Team
Argonne Leadership Computing Facility

Aug 11, 2023

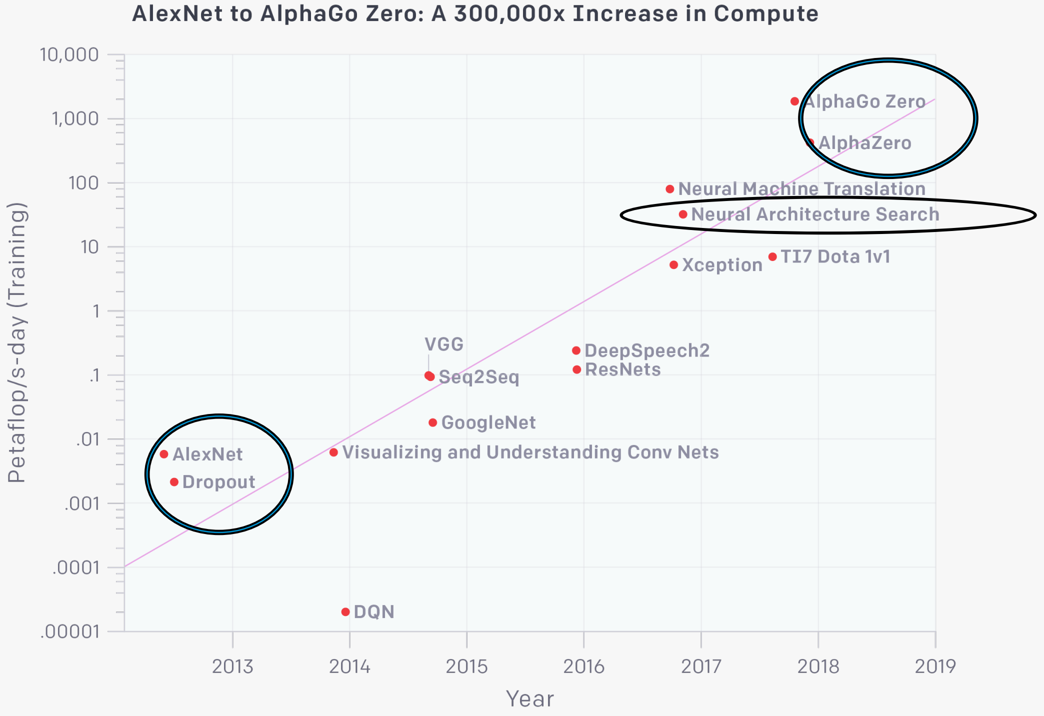
huihuo.zheng@anl.gov

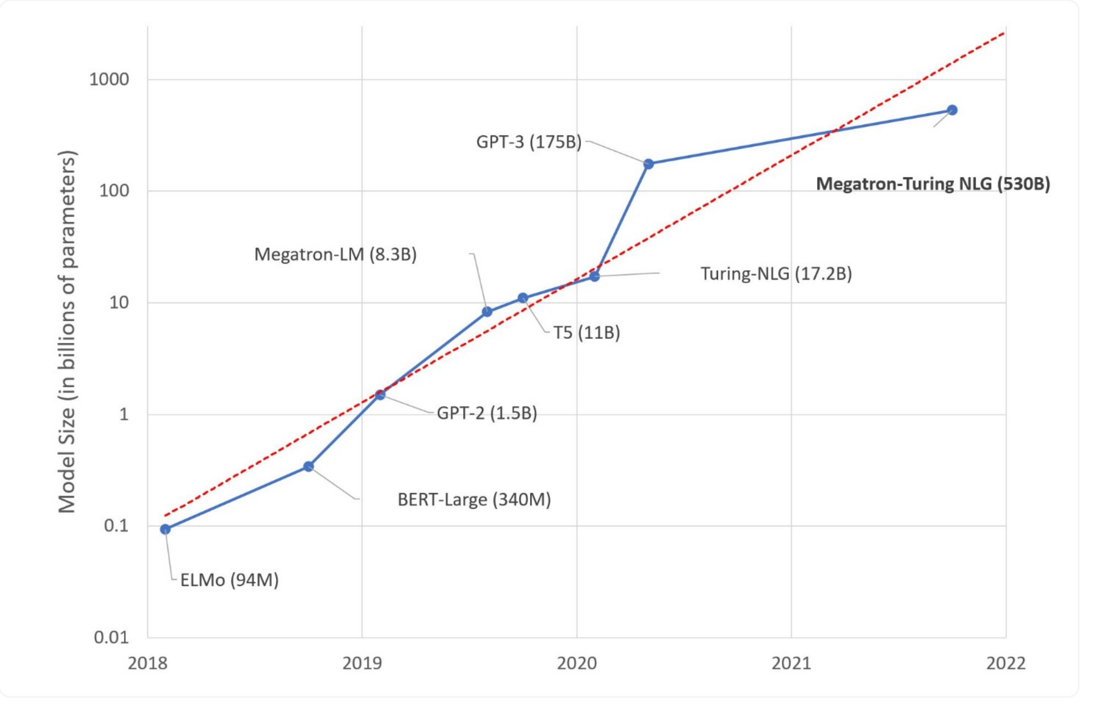
Outline

- The need for distributed training
- State-of-the-art parallelization schemes
- Data parallel training in details
- I/O and data management in distributed training
- Hands on

The need for distributed training on HPC

“Since 2012, the amount of compute used in the largest AI training runs has been increasing exponentially with a 3.5 month doubling time (by comparison, Moore’s Law had an 18 month doubling period).”





https://openai.com/blog/ai-and-compute/

Large language model: # parameters grows by about 10x every year

Distributed deep learning for ResNet-50

TRAINING TIME AND TOP-1 VALIDATION ACCURACY WITH RESNET-50 ON IMAGENET

		Batch Size	Processor	DL Library	Time	Accuracy
He et al. [1]	2016	256	Tesla P100 × 8	Caffe	29 hours	75.3 %
Goyal et al. [2]		8,192	Tesla P100 × 256	Caffe2	1 hour	76.3 %
Smith et al. [3]		8,192 → 16,384	full TPU Pod	TensorFlow	30 mins	76.1 %
Akiba et al. [4]		32,768	Tesla P100 × 1,024	Chainer	15 mins	74.9 %
Jia et al. [5]		65,536	Tesla P40 × 2,048	TensorFlow	6.6 mins	75.8 %
Ying et al. [6]		65,536	TPU v3 × 1,024	TensorFlow	1.8 mins	75.2 %
Mikami et al. [7]		55,296	Tesla V100 × 3,456	NNL	2.0 mins	75.29 %
Yamazaki et al	2019	**81,920**	**Tesla V100 × 2,048**	**MXNet**	**1.2 mins**	**75.08%**

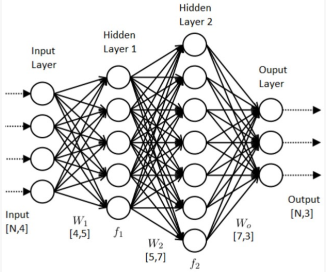
Quoted from Masafumi Yamazaki, arXiv:1903.12650

Training Large Natural Language Model is expensive

Scheme	Number of parameters (billion)	Model-parallel size	Batch size	Number of GPUs	Microbatch size	Achieved teraFlOP/s per GPU	Training time for 300B tokens (days)
ZeRO-3 without Model Parallelism	174.6	1	1536	384	4	144	90
				768	2	88	74
				1536	1	44	74
	529.6	1	2560*	640	4	138	169
			2240	1120	2	98	137
				2240	1	48	140
PTD Parallelism	174.6	96	1536	384	1	153	84
				768	1	149	43
				1536	1	141	23
	529.6	280	2240	560	1	171	156
				1120	1	167	80
				2240	1	159	42

Narayanan, D et al. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; ACM: St. Louis Missouri, 2021; pp 1–15.

The need for distributed training

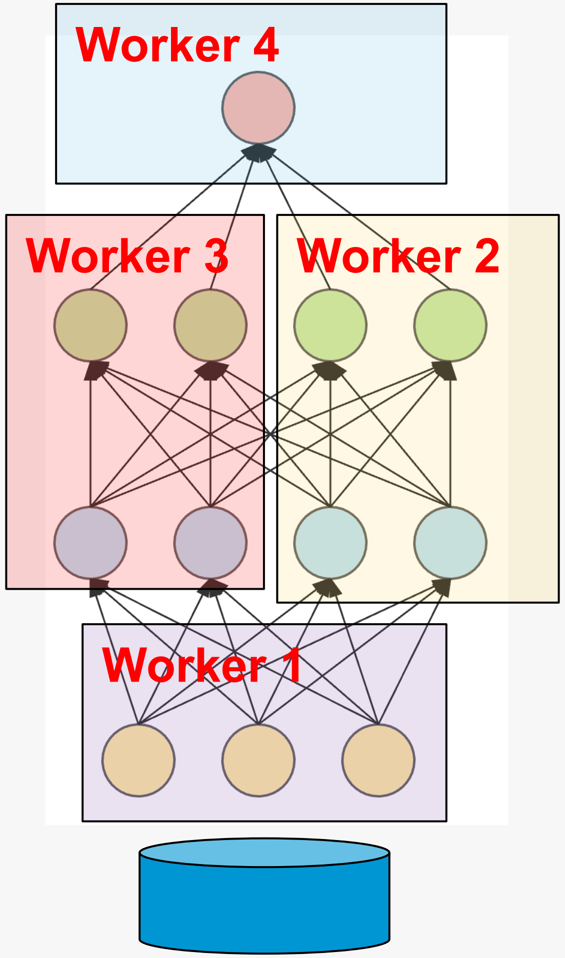


- Increase of model complexity leads to dramatic increase of the amount of computation;
- Increase of the size of dataset makes sequentially scanning the whole dataset increasingly impossible;
- Coupling of deep learning to traditional HPC simulations might require distributed training and inference.

Examples of scientific large scale deep learning

- Thorsten Kurth, Exascale Deep Learning for Climate Analytics, arXiv:1810.01993 (Gordon Bell Prize)
- R. M. Patton, Exascale Deep Learning to Accelerate Cancer Research, arXiv:1909.1229
- N. Laanait, Exascale Deep Learning for Scientific Inverse Problems, arXiv:1909.11150
- W. Dong et al, Scaling Distributed Training of Flood-Filling Networks on HPC Infrastructure for Brain Mapping, arXiv:1905.06236
- A Khan, et al, Deep learning at scale for the construction of galaxy catalogs in the Dark Energy Survey Physics Letters B 795, 248-258
- Narayanan, D.; et al, Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. arXiv: 2104.04473

Parallelization schemes – Model Parallelism (MP)



Model parallelism

```
import torch
import torch.nn as nn
import torch.optim as optim


class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

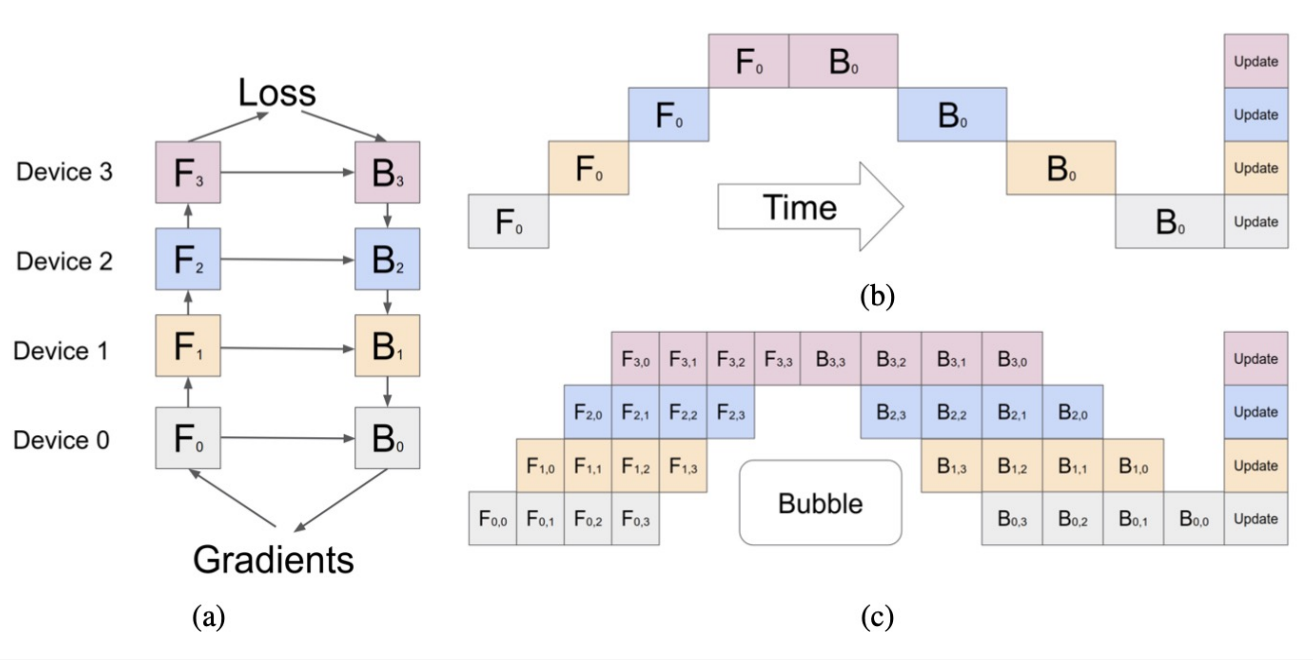
```
model = ToyModel()
loss_fn = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)

optimizer.zero_grad()
outputs = model(torch.randn(20, 10))
labels = torch.randn(20, 5).to('cuda:1')
loss_fn(outputs, labels).backward()
optimizer.step()
```

PyTorch multiple GPU model parallelism within a node

https://pytorch.org/tutorials/intermediate/model_parallel_tutorial.html

Parallelization schemes – Pipepline parallelism (PP)

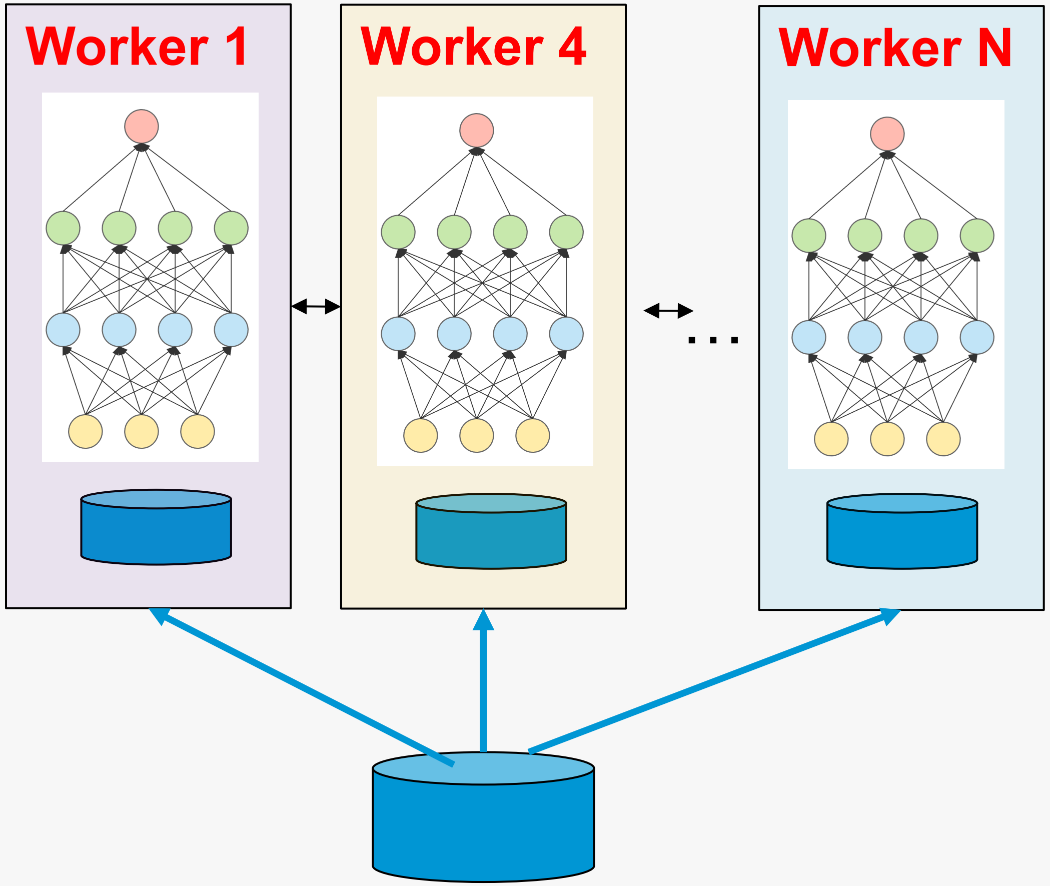


Pipeline libraries:

- GPipe: arXiv:1811.06965
- Pipe-torch:
  DOI: 10.1109/CBD.2019.00020
- PipeDream: arXiv:1806.03377
- HetPipe: arXiv:2005.14038
- DAPPLE: arXiv:2007.01045
- PyTorch Distributed RPC Frameworks:
  https://pytorch.org/tutorials/intermediate/dist_pipeline_parallel_tutorial.html
- DeepSpeed:
  https://github.com/microsoft/DeepSpeed

- Partition model layers into multiple groups (stages) and place them on a set of inter-connected devices.
- Each input batch is further divided into multiple micro-batches, which are scheduled to run over multiple devices in a pipelined manner.

Parallelization schemes – Data Parallelism (DP)



Data parallelism

- Model is replicated on each worker
- Each worker processes a subset of the minibatch
- Sync up the weights before updating the model

Popular frameworks support DP

DDP

HOROVOD

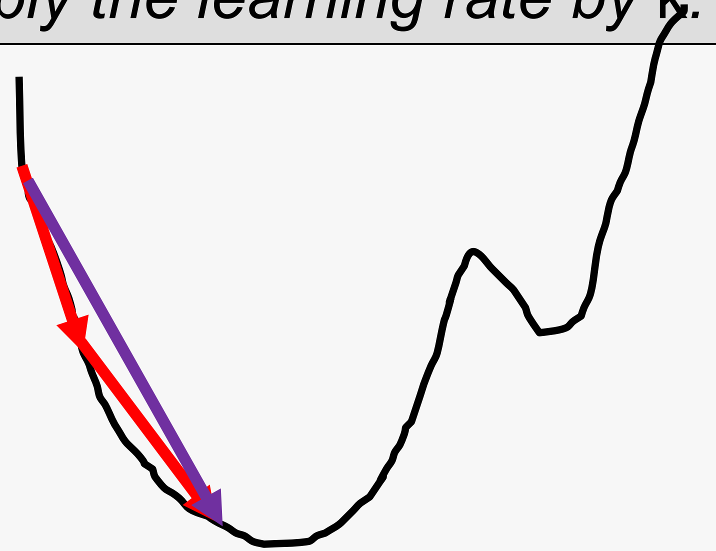
DeepSpeed

https://leimao.github.io/blog/PyTorch-Distributed-Training/
https://github.com/horovod/horovod
https://github.com/microsoft/DeepSpeed

Data parallel training in details

Linear rate scaling rule

When the minibatch size is multiplied by k, multiply the learning rate by k.



Mini-batch stochastic Gradient Descent

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Learning rate, lr   Mini-batch

$$lr_{scale} = lr * nprocs$$

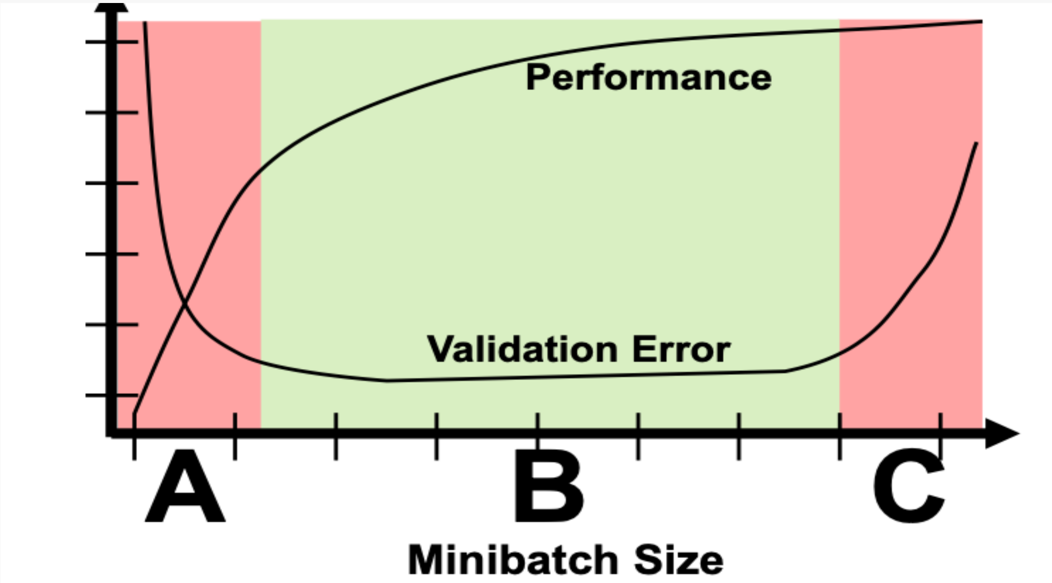
Typical practice / suggestion:

- keep local batch size per worker, i.e., increase the global batch size linearly
- Increase the learning rate proportionally

Actuality: need to adjust the batch size and learning rate at different scale

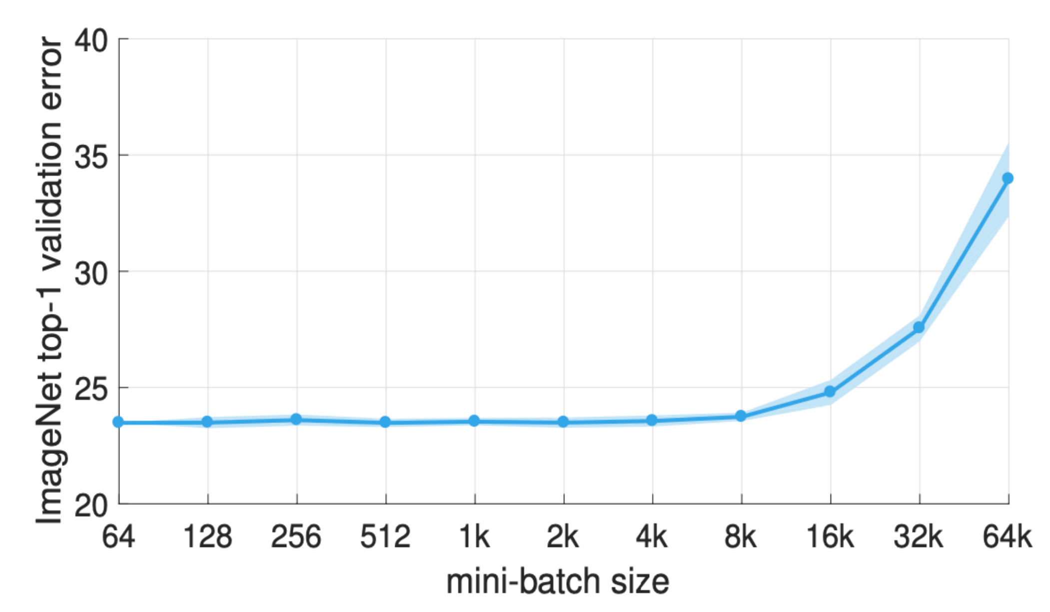
S. McCandlish, J. Kaplan, D. Amodei, arXiv:1812.06162

Potential issues for large batch size and learning rate



Minibatch Size Effect on Accuracy and Performance

Tal Ben-Nun and Torsten Hoefler, arXiv:1802.09941



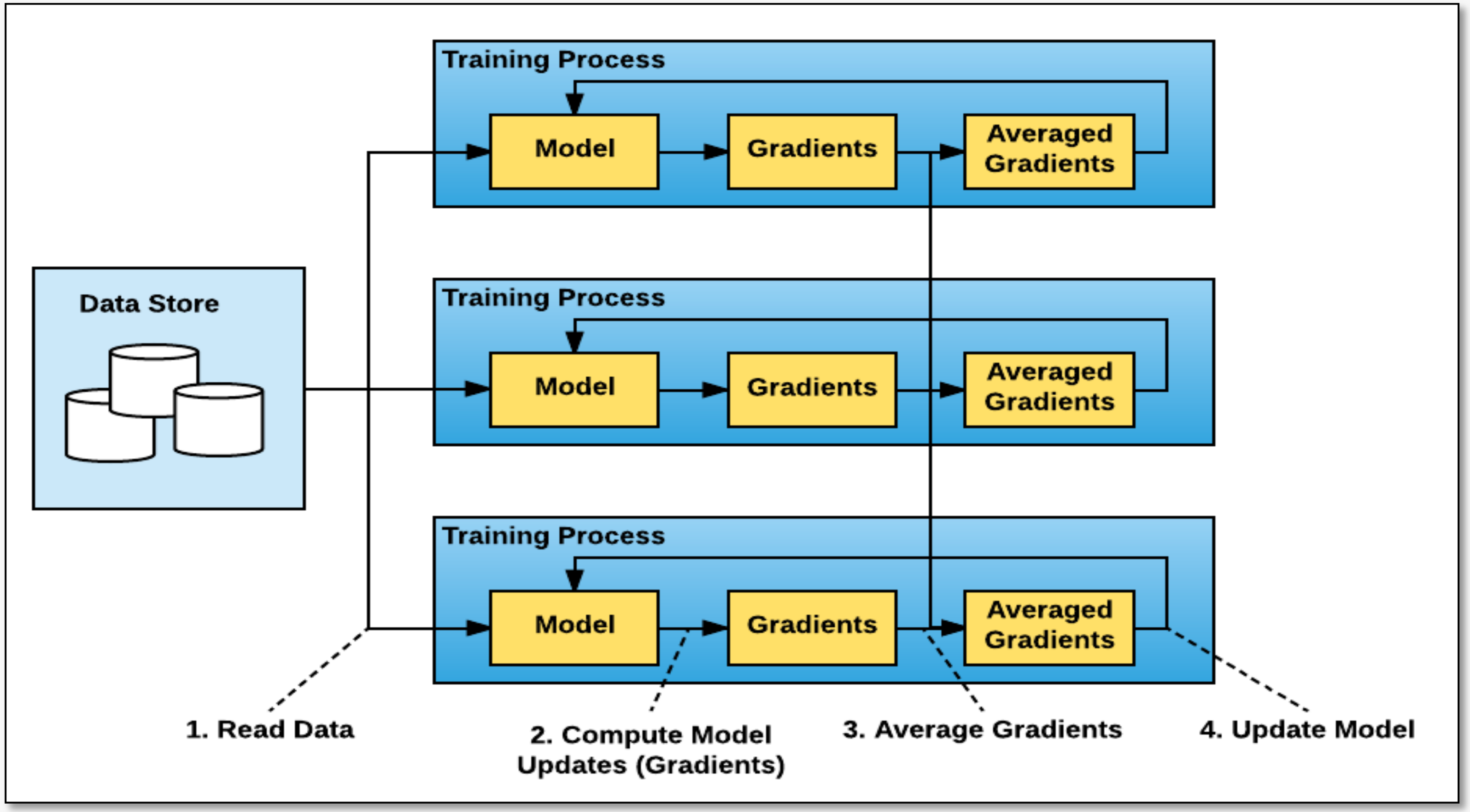
Validation error for different mini-batch size for Resnet50

P. Goyal et al,arXiv: 1706.02677

- Stability of optimization – can be solved by learning rate warming up
- Generation gap – systematic issue; has to reduce the learning rate

Generation gap: Keskar et al, arXiv:1609.04836

Data parallel training with Horovod



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

Data parallel training with Horovod

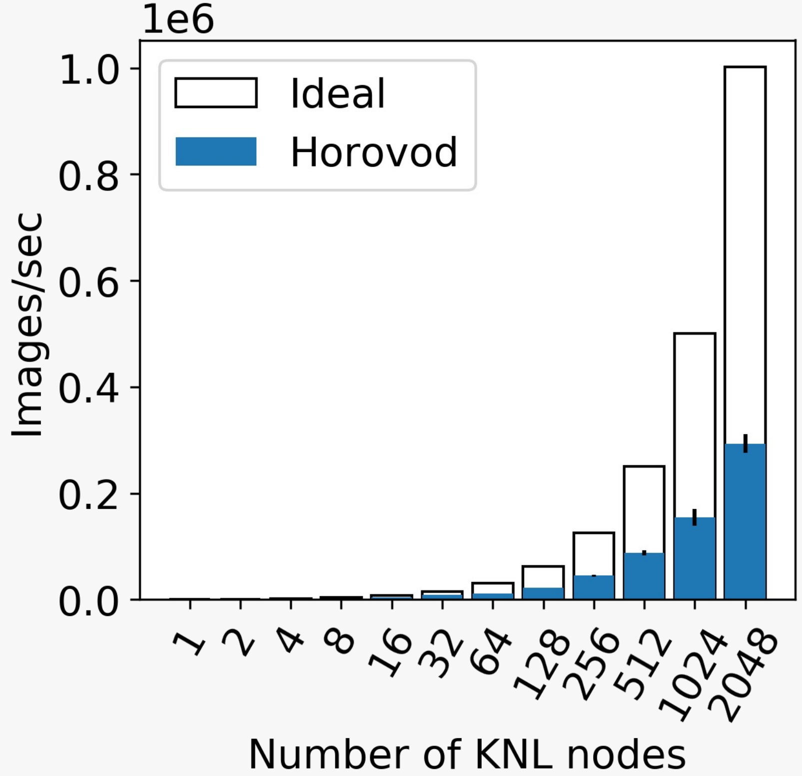
Steps to parallelize a series code:

- Import Horovod modules and initialize horovod
- Scale the learning rate by number of workers
- Wrap optimizer in hvd.DistributedOptimizer
- Broadcast the weights from worker 0 to all the workers
- Worker 0 saves the check point files
- Dataset sharding: make sure different workers load different samples.

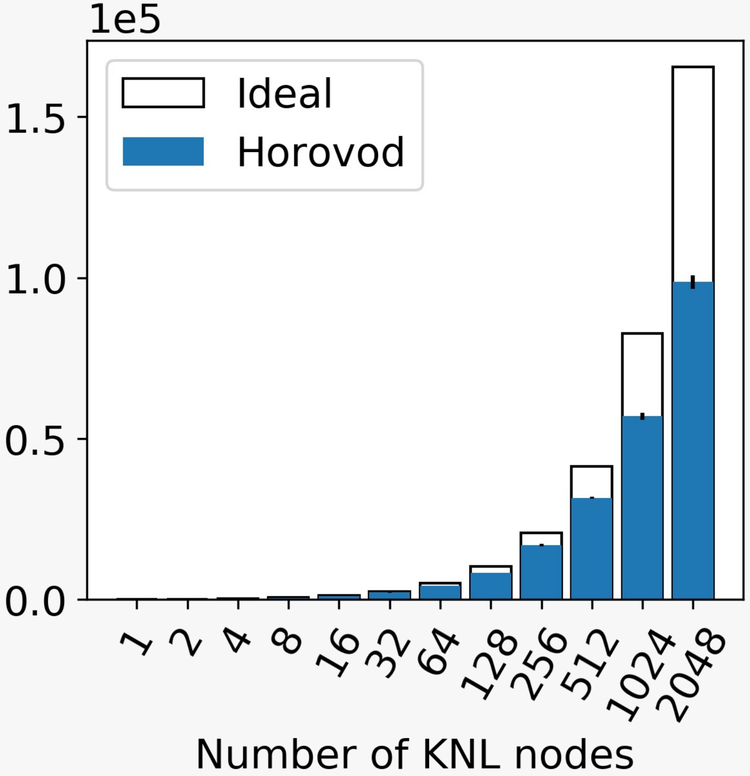
Instruction on how to change the code is here
Tensorflow: 04_keras_cnn_verbose_hvd.py
Pytorch: 04_pytorch_cnn_hvd.py

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

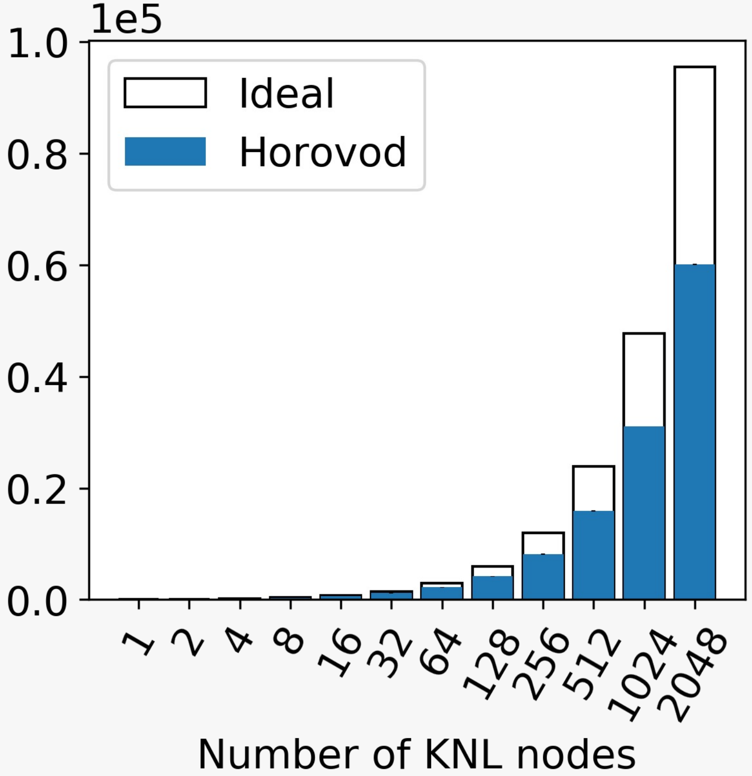
Scaling TensorFlow using Data parallelism on Theta @ ALCF: fixing local batch size = 512



AlexNet

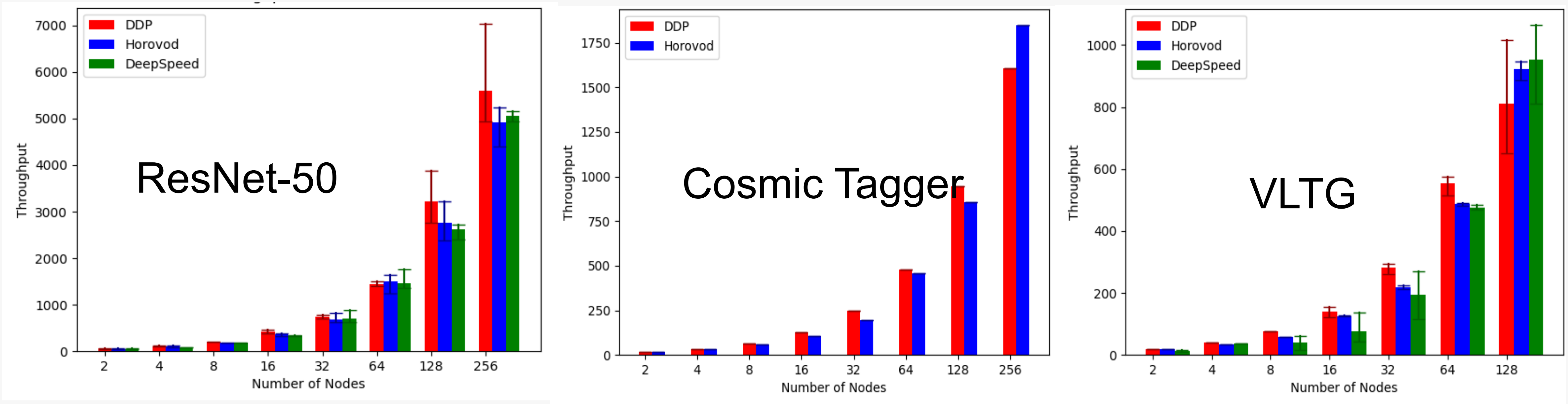


ResNet-50



Inception V3

Different frameworks show similar scaling efficiency



Horovod, DDP, and DeepSpeed show similar performance for three PyTorch models. Evaluation were done on Polaris @ ALCF

Zhenhao Z.
@IIT

https://github.com/argonne-lcf/dl_scaling.git

Argonne NATIONAL LABORATORY

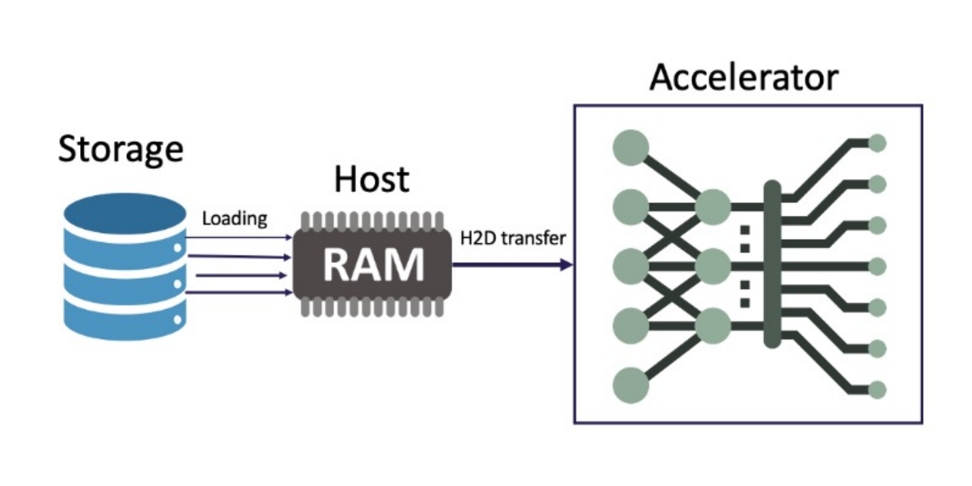
Data Management and I/O for AI

Devarajan, H.; Zheng, H.; Kougkas, A.; Sun, X.-H.; Vishwanath, V. DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*; 2021; pp 81–91.

DLIO Benchmark: https://github.com/argonne-lcf/dlio_benchmark.git

MLPerf Storage: https://mlcommons.org/en/news/mlperf-storage/

Deep Learning I/O characteristics



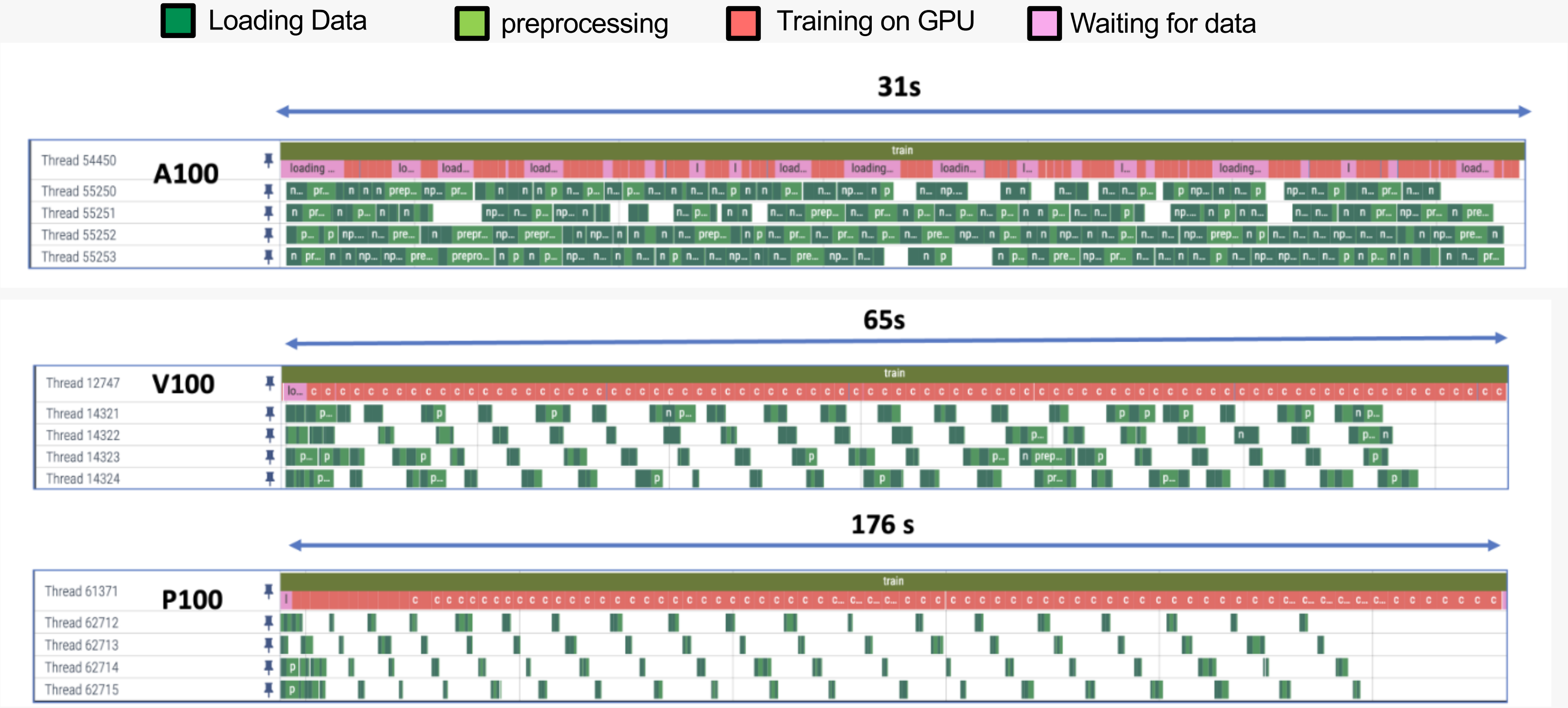
Typical process of AI training. The dataset is loaded from the storage to the host RAM and then feed into the accelerators for training.

Characteristics of I/O for AI applications

- Read intensive
- Metadata intensive
- Small and sparse I/O operations
- Random access
- Complex data format (json, text, key-value store)
- Utilizing storage hierarchy
- Multithreading background I/O

Huihuo Zheng and Venkatram Vishwanath, Data Management for Scientific Artificial Intelligence Workloads, ASCR Data workshop, 2022

I/O bottleneck for UNet3D workload on fast accelerator

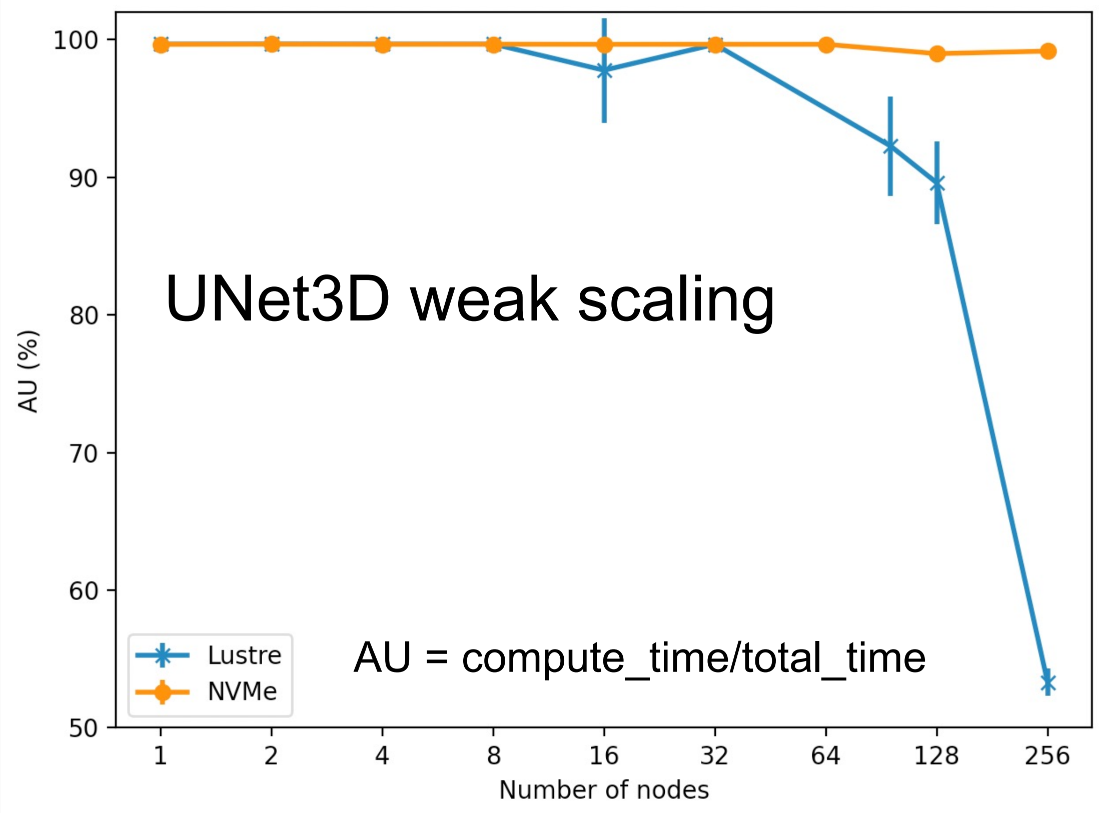


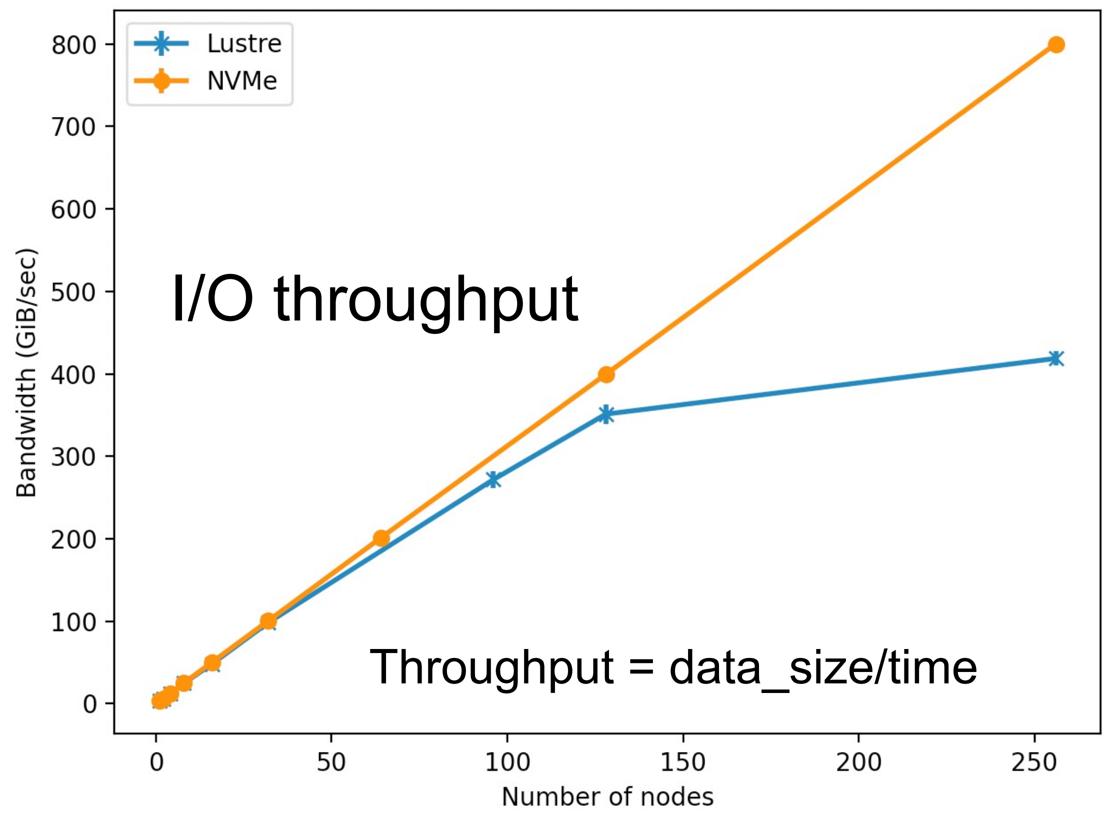
Timeline tracing for training the UNet3D workload on a single GPU on JLSE with GPFS file system. This shows that I/O become a bottleneck for faster accelerator

UNet3D Model: https://github.com/mlcommons/training/tree/master/image_segmentation/pytorch

https://github.com/argonne-lcf/dlio_benchmark.git

Scaling bottleneck from IO for UNet3D workload (simulated using DLIO Benchmark)

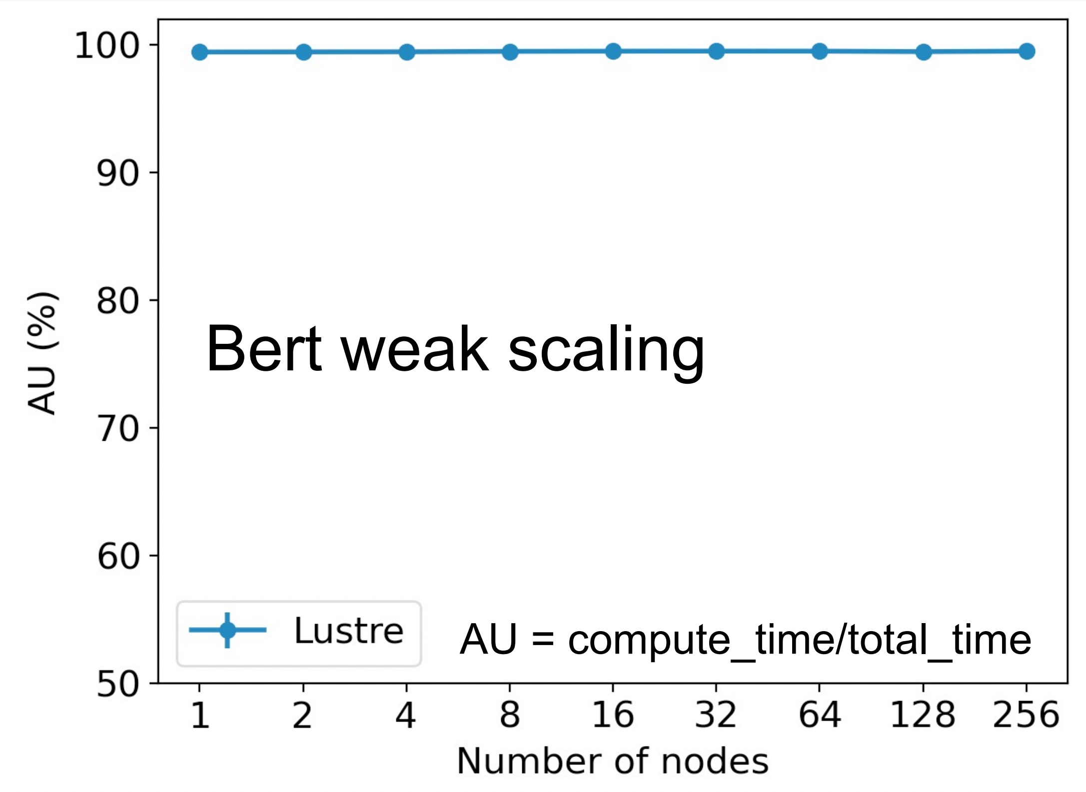


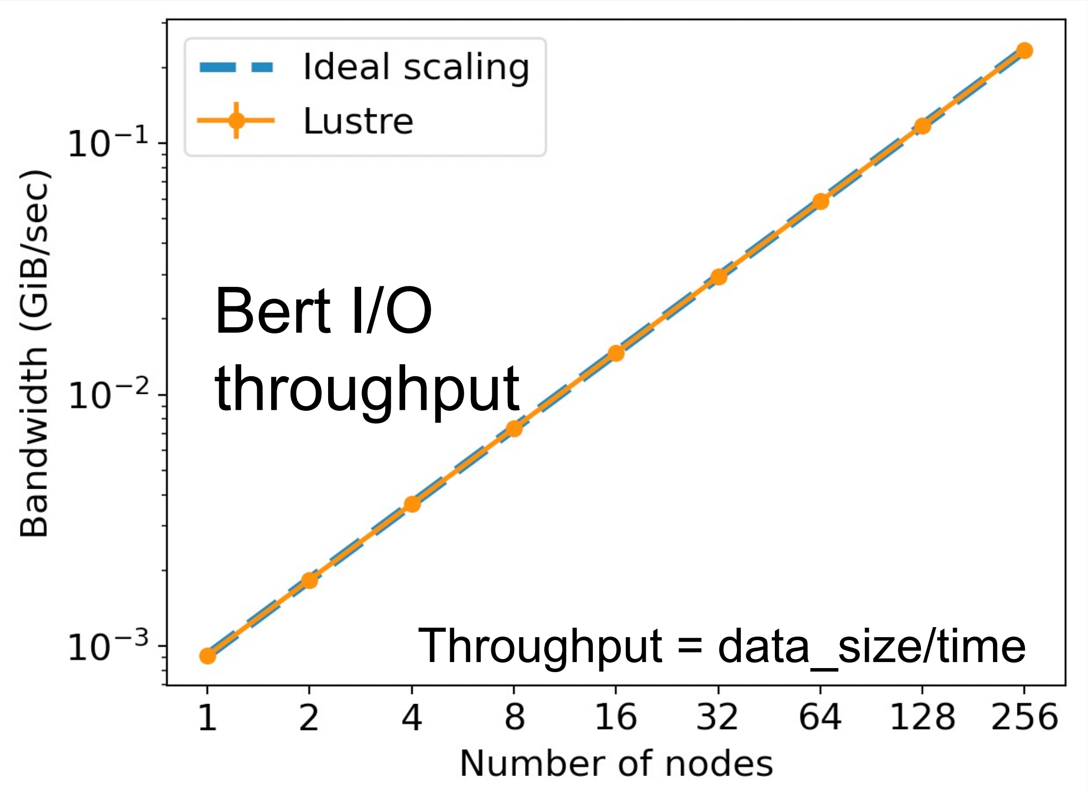


Accelerator utilization (AU) and I/O throughput at different scale on Polaris for UNet3D model, with Lustre file system and NVMe -> staging helps

https://github.com/argonne-lcf/dlio_benchmark.git

Ideal scaling for less I/O intensive workload – Bert (simulated using DLIO Benchmark)





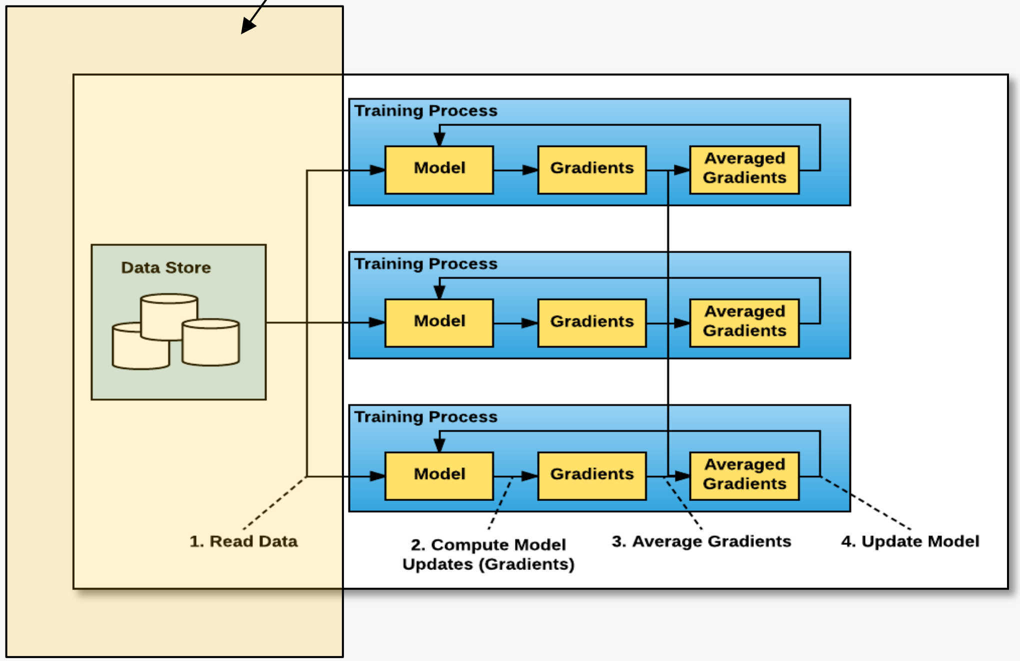
Accelerator utilization (AU) and I/O throughput at different scale on Polaris for Bert model, with the Lustre file system

https://github.com/argonne-lcf/dlio_benchmark.git

Tips for I/O and data management

- Preprocess the raw data (resize, interpolation, etc) into binary format before the training;
- Store the dataset in a reasonable way (file per sample, single shared file, or multiple samples per file)
- Optimal setting (Lustre stripe count, size)
- Remember to shard the dataset;
- Prefetch and caching the data (from disk; from host to device; staging to NVMe, SSDs);
- Use more I/O workers to load data concurrently (e.g., adjust num_workers in TorchDataLoader)

I/O and data management



Streaming I/O using Data Loader

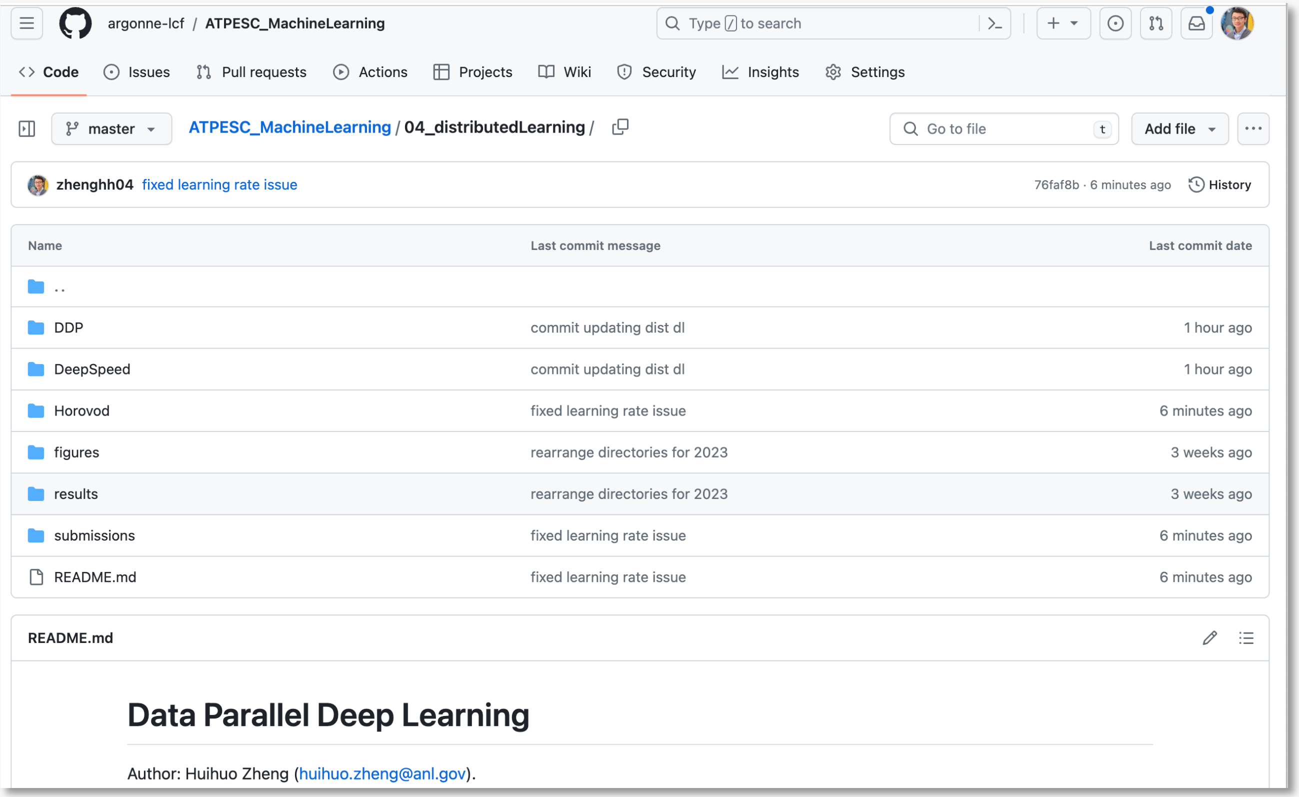
- TensorFlow Data Pipeline
- PyTorch Data Loader
- Nvidia Dali Data Loader

Main take aways

- Distributed training can be done through model parallelism and Data parallelism.
- Data parallelism frameworks: Horovod, DDP (PyTorch only), DeepSpeed (Pytorch only) → similar performance.
- Efficient data management and I/O is crucial for data intensive training

Hands on

$ git clone git@github.com:argonne-lcf/ATPESC_MachineLearning.git
$ cd ATPESC_MachineLearning/04_distributedLearning
$ qsub submissions/qsub_thetagpu.sc or qsub submissions/qsub_polaris.sc



Thank you!

huihuo.zheng@anl.gov