Distributed Deep Learning

Huihuo Zheng

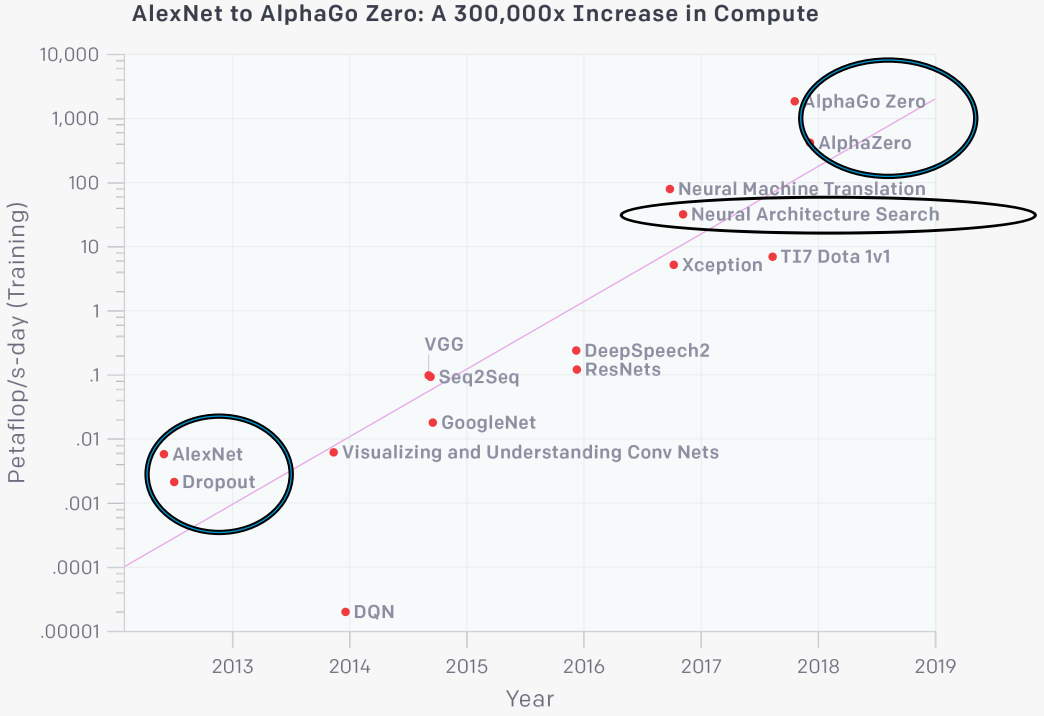
Argonne Leadership Computing Facility
Argonne National Laboratory

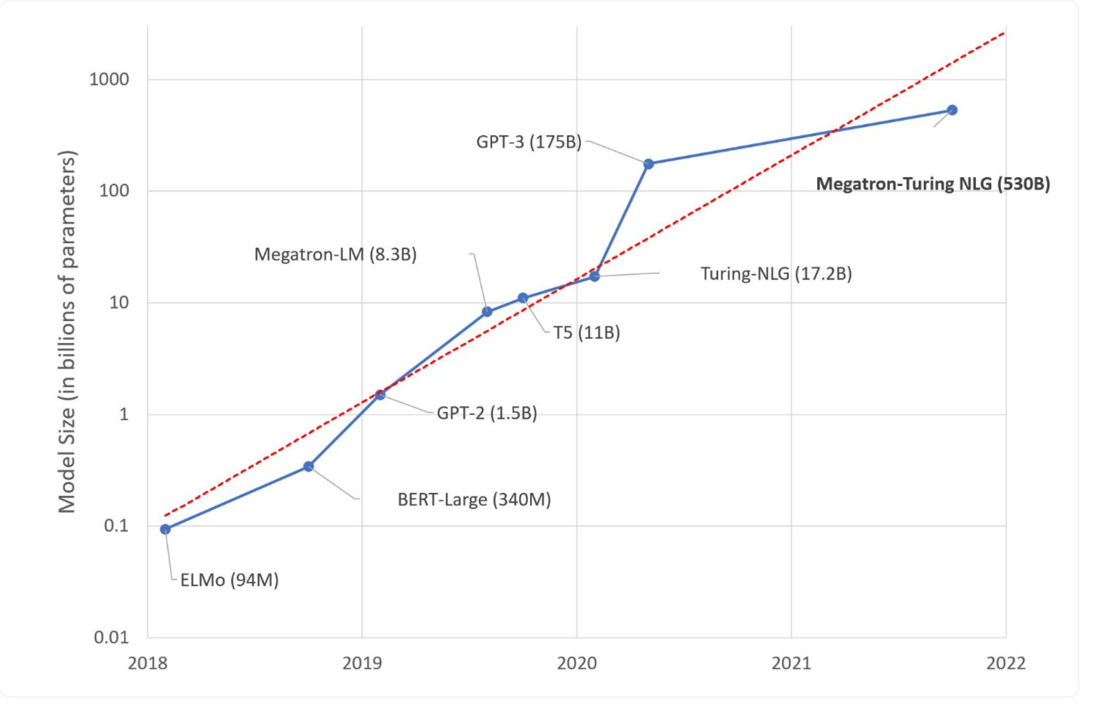
October 11, 2023

huihuo.zheng@anl.gov

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

Training Large Natural Language Model is expensive

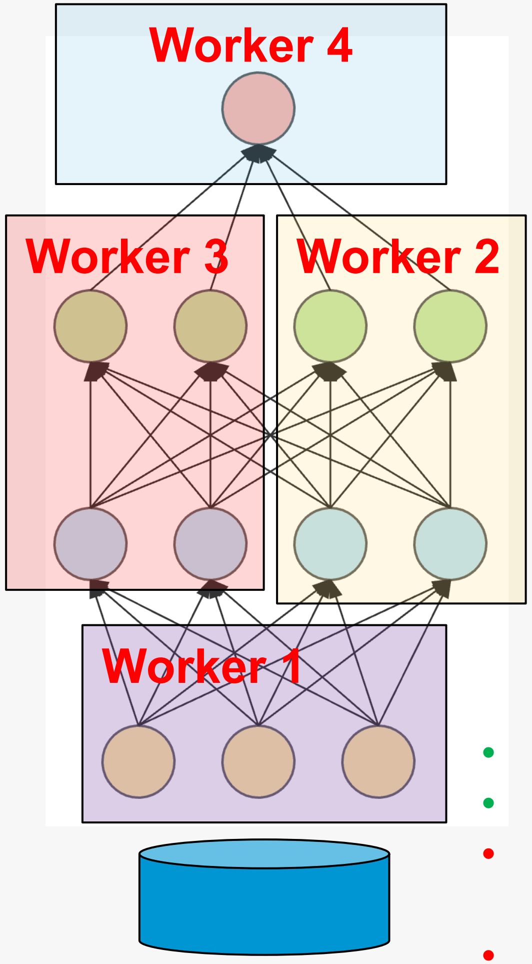
Scheme	Number of parameters (billion)	Model-parallel size	Batch size	Number of GPUs	Microbatch size	Achieved teraFlOP/s per GPU	Training time for 300B tokens (days)
ZeRO-3 without Model Parallelism	174.6	1	1536	384	4	144	90
				768	2	88	74
				1536	1	44	74
	529.6	1	2560*	640	4	138	169
			2240	1120	2	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.

Outline

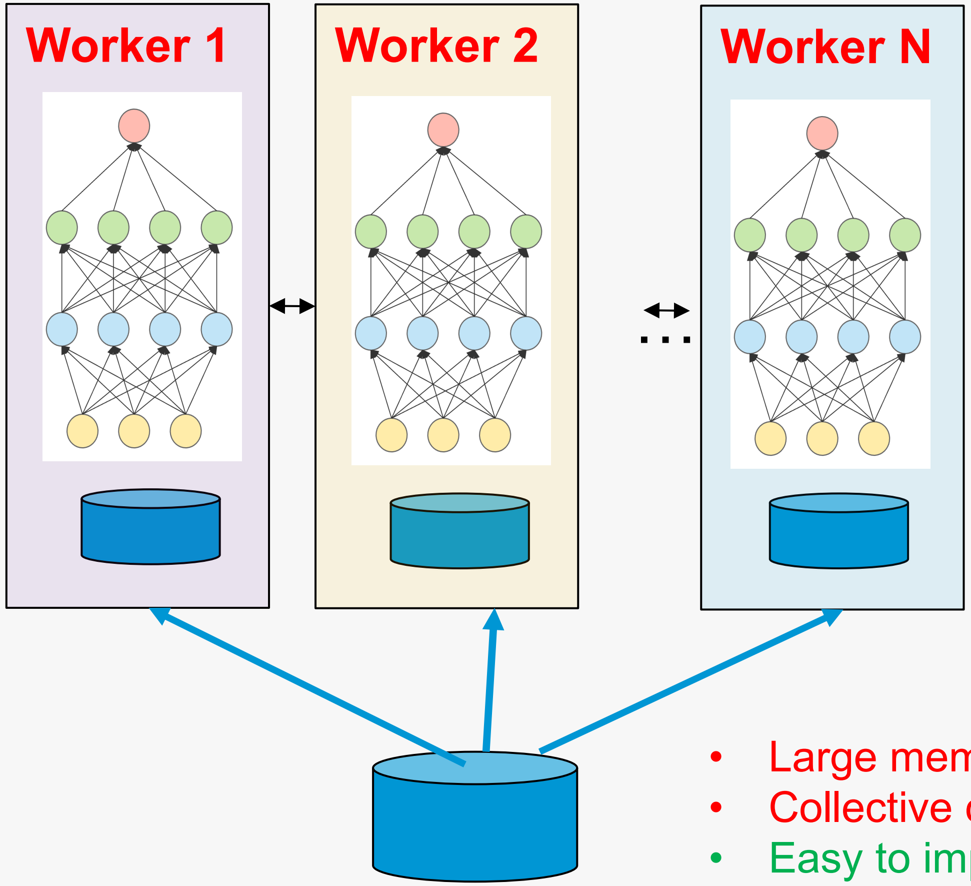
- Parallelization Schemes
- Distributed Training Frameworks
- I/O and Data Management

Parallelization schemes



- Large model
- Local communication
- Not standard way of distributing
- Load imbalance

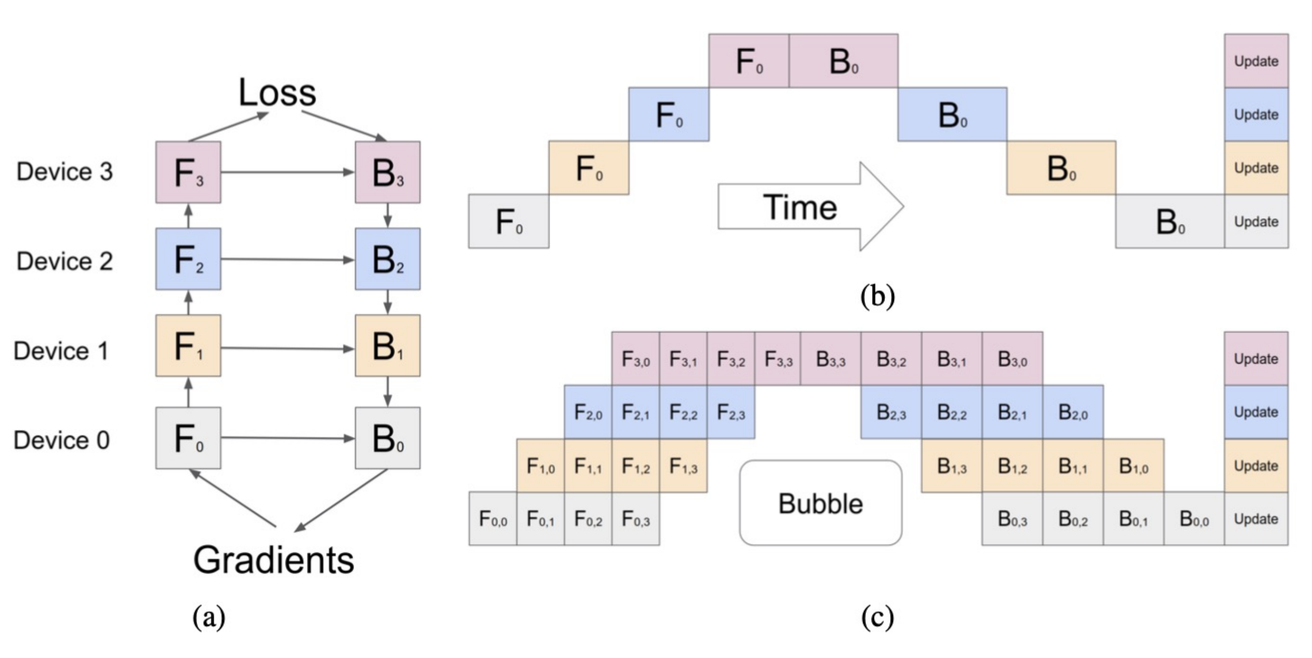
Model parallelism



Data parallelism

- Large memory footprint
- Collective communication
- Easy to implement
- Load balance

Parallelization schemes – Pipeline 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.

Distributed Training Frameworks

- TensorFlow
- PyTorch
- Keras
- MXNet

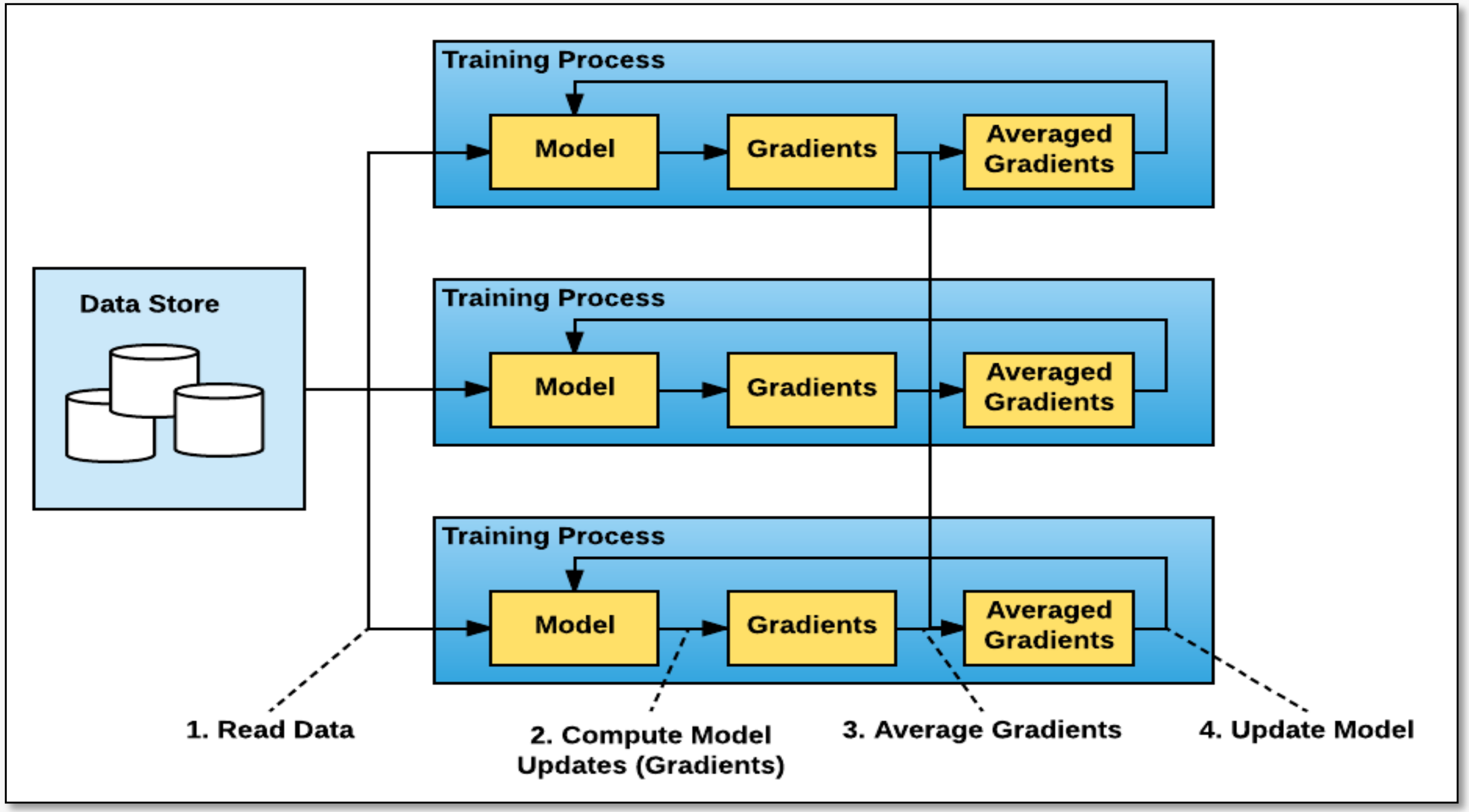
PyTorch

PyTorch

```
$ module load conda/2023-10-04
$ conda activate
```

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

Data parallel training



HOROVOD

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

Horovod

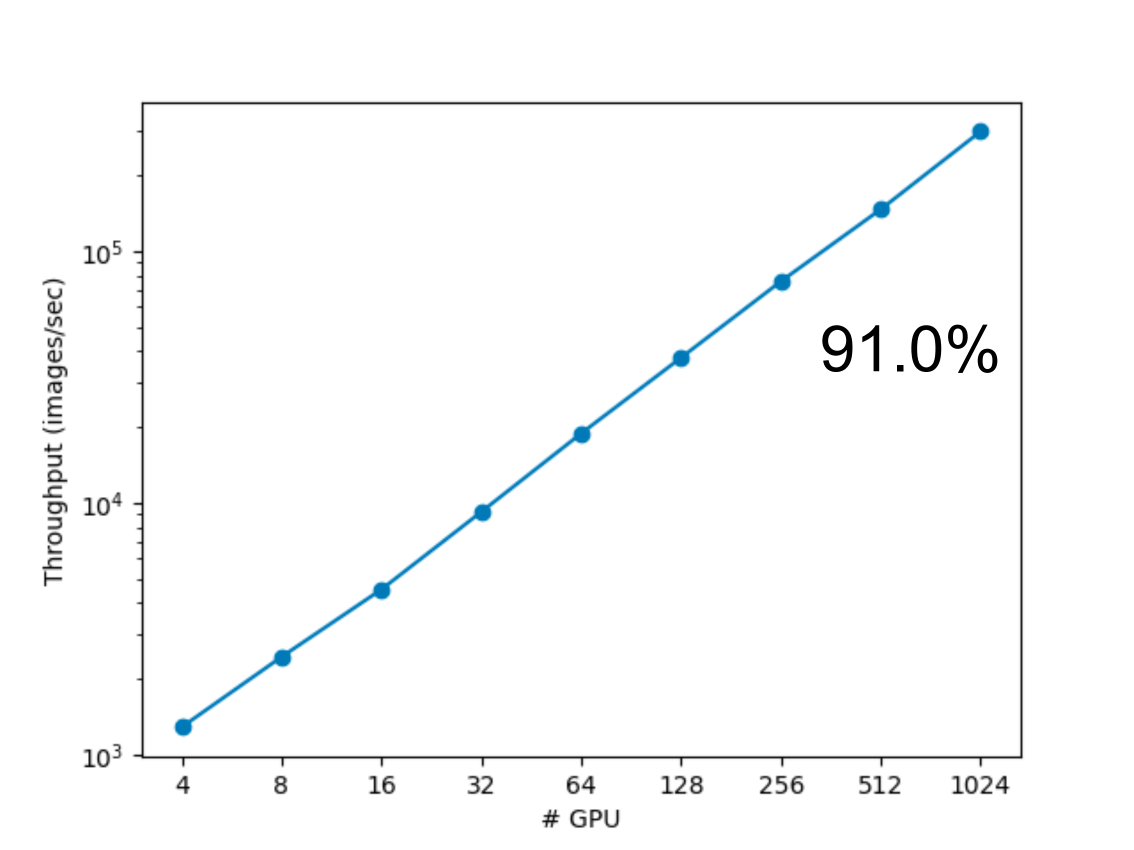
- Import Horovod modules and initialize horovod
- Scale the learning rate by number of workers
- Wrap optimizer in hvd.DistributedOptimizer
- Broadcast the weights from worker 0 to all the workers
- Worker 0 saves the check point files
- Dataset sharding: make sure different workers load different samples.

Instruction on how to change the code is here
Tensorflow: keras_cnn_verbose_hvd.py
Pytorch: pytorch_cnn_hvd.py

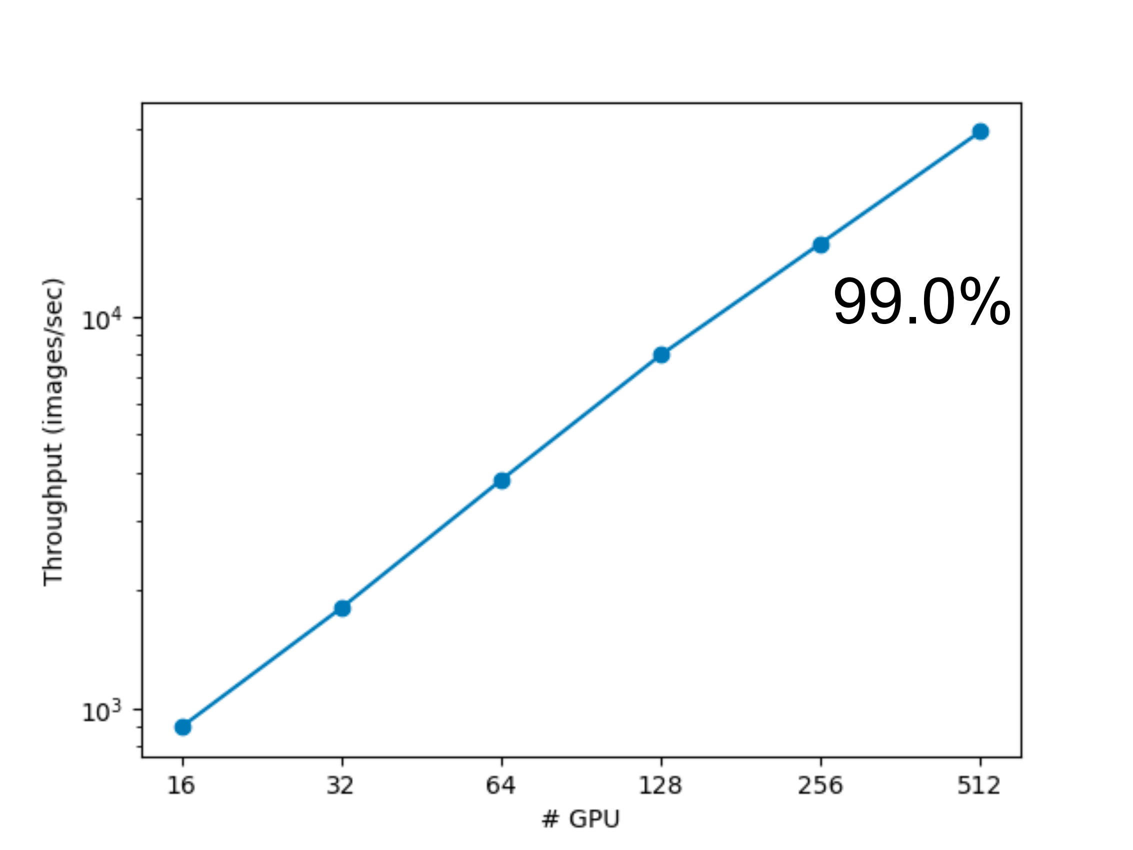
General practices

- Scale global batch size and learning rate in proportional to number of workers
- A few warm up epochs with smaller learning rate to stabilize the training
- Adjust learning rate (and/or other hyperparameters) according to convergence behavior (different scales have different behavior)
- Avoiding averaging metrics on each training step. Only do averaging at the end of each epoch

Scaling of Training Throughput on Polaris

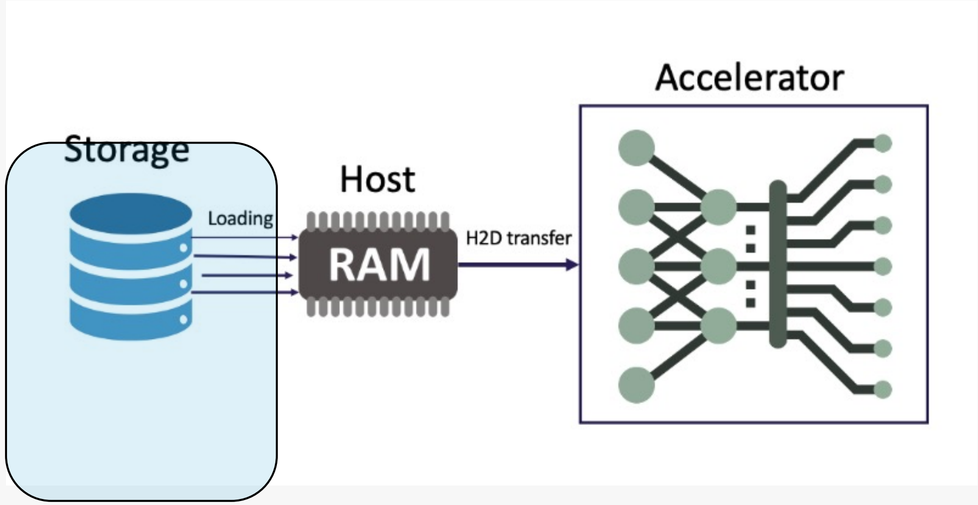


ResNet50 (TF+HVD)



CosmoFlow (PT+DDP)

Data Management and I/O



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

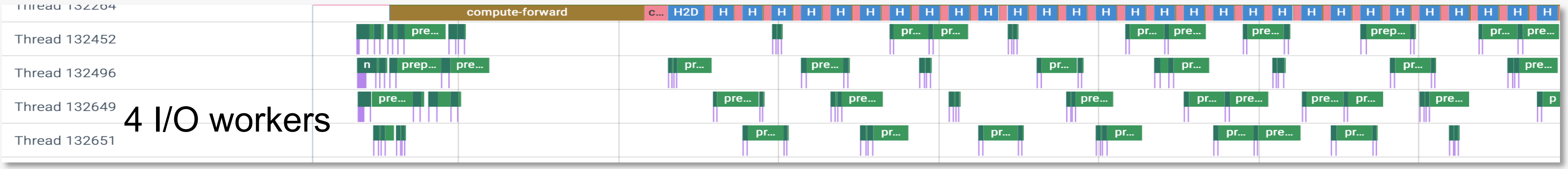
Devarajan, H.; Zheng, H.; Kougkas, A.; Sun, X.-H.; Vishwanath, V. DLIO: A Data-Centric Benchmark for Scientific Deep Learning Applications. *(CCGrid*; 2021; pp 81–91.
DLIO Benchmark: https://github.com/argonne-lcf/dlio_benchmark.git
MLPerf Storage: https://mlcommons.org/en/news/mlperf-storage/

I/O Tracing for UNet3D workload

Loading Data   preprocessing   Waiting for data   Training on GPU

0 I/O workers

90s per epoch



4 I/O workers

19s per epoch

Timeline tracing for training the UNet3D workload on Polaris

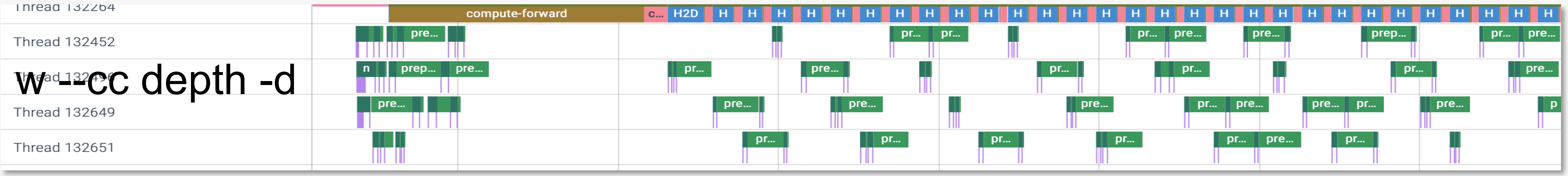
- Multi-threading allowing overlap of compute and I/O

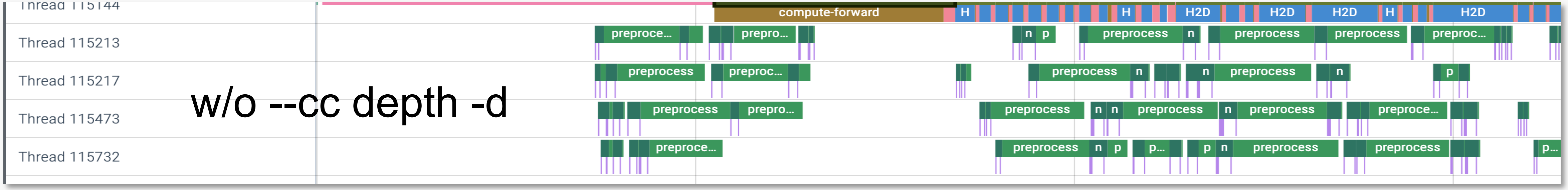
UNet3D Model: https://github.com/mlcommons/training/tree/master/image_segmentation/pytorch

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

I/O Tracing for UNet3D workload

Loading Data preprocessing Waiting for data Training



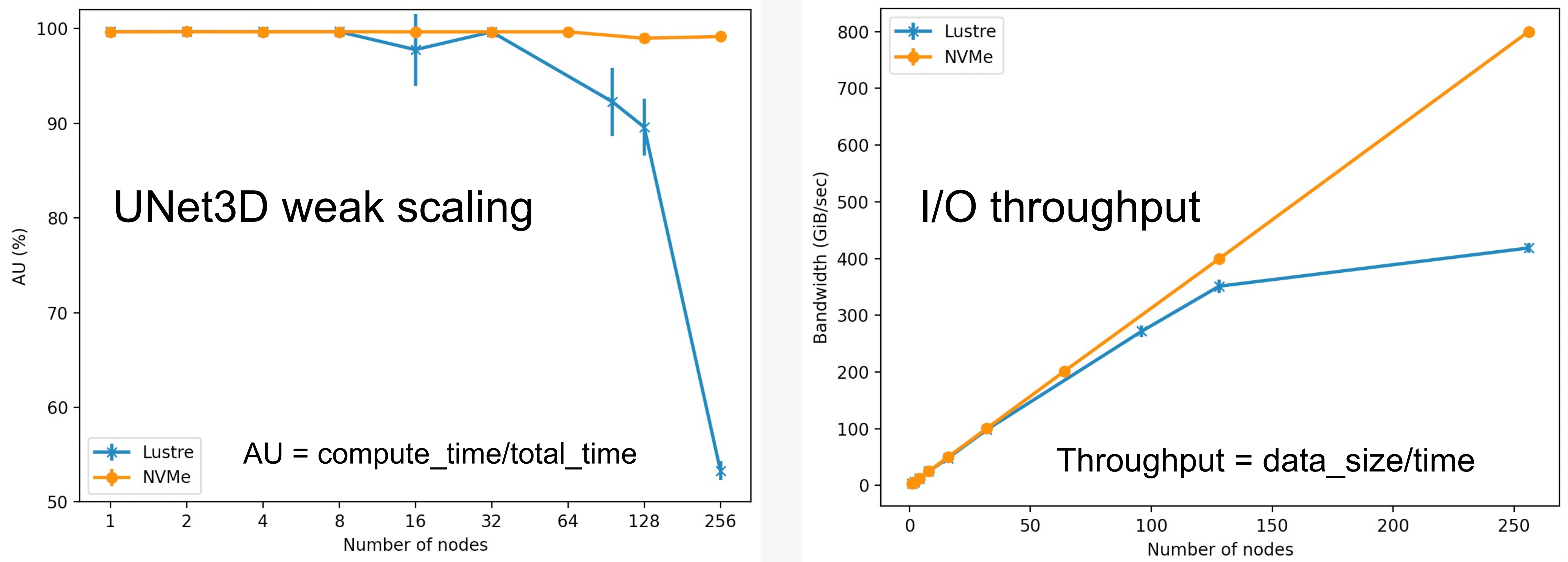


- CPU binding is crucial for preprocessing

	w/ --cc depth –d	w/o --cc depth –d
I/O	6s	30s
Preprocess	19s	89s

Argonne NATIONAL LABORATORY

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

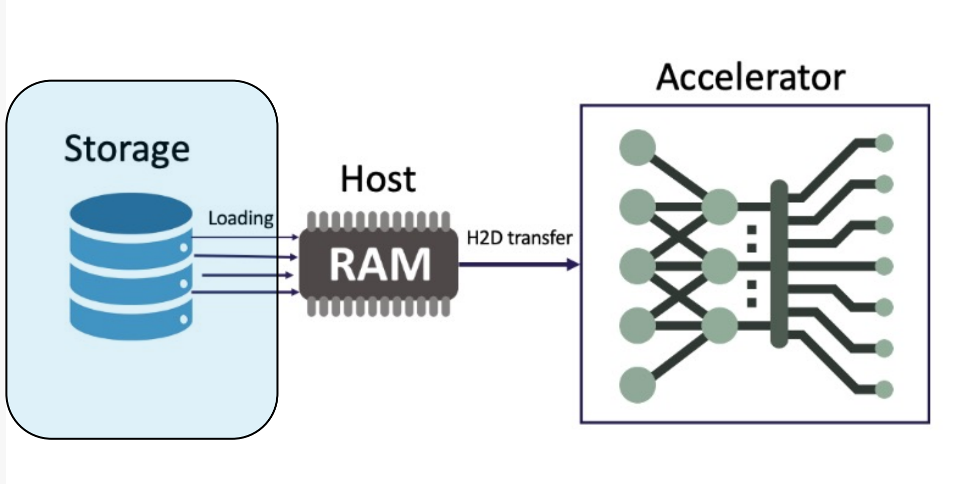


Accelerator utilization (AU) and I/O throughput at different scale on Polaris for UNet3D model, with Lustre file system and NVMe -> staging helps

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

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)



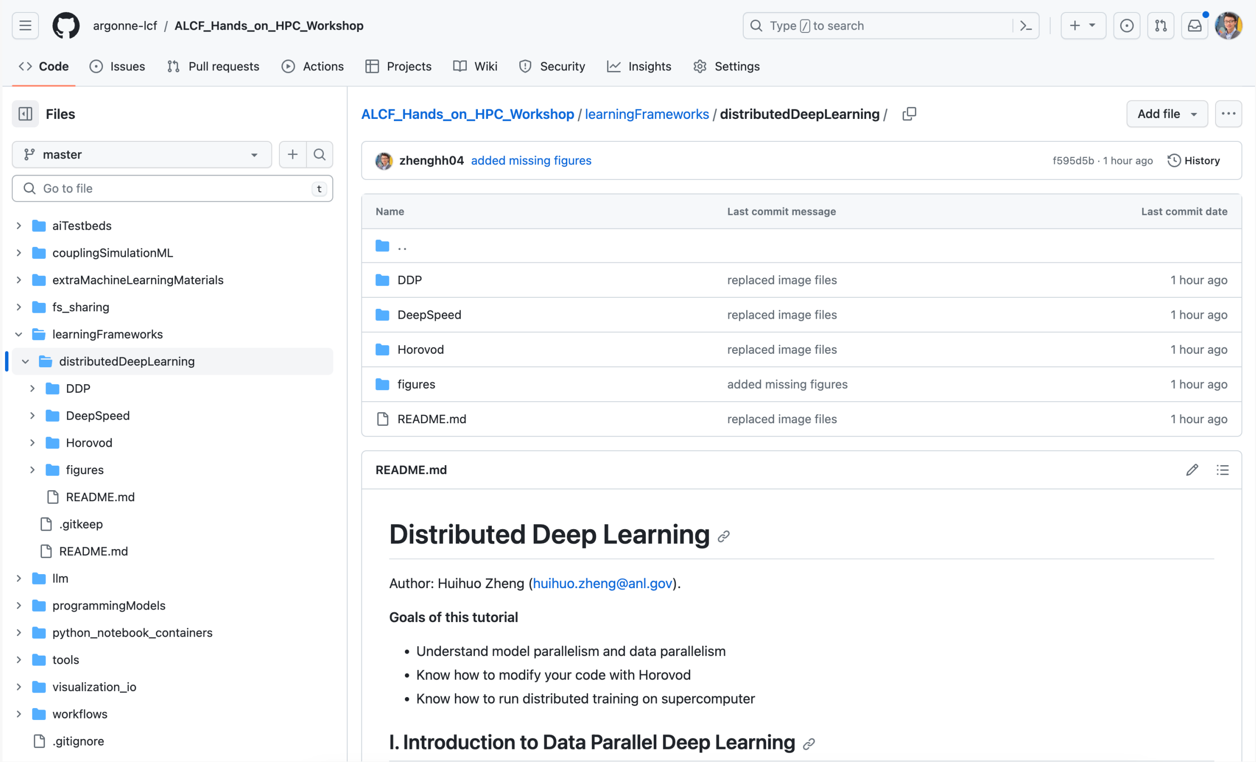
Streaming I/O using Data Loader

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

Current issue on Polaris: TorchDataLoader num_workers>0 will cause hang on multiple nodes→ use Dali Data Loader instead

Hands on

$ git clone git@github.com:argonne-lcf/ALCF_Hands_on_HPC_Workshop.git
$ cd ALCF_Hands_on_HPC_Workshop/**learningFrameworks/distributedDeepLearning**
$ cd Horovod/; qsub qsub_polaris.sc



Acknowledgments

- This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.
- We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.

Thank you!

huihuo.zheng@anl.gov