Deep Learning at Scale on Perlmutter

NERSC Data Day
Feb 21st, 2024

Peter Harrington
Data & AI Services Group
NERSC

The Deep Learning revolution

2M pixels, 60 frames per second, one minute long!

https://openai.com/sora

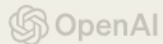
Prompt: A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She walks confidently and casually. The street is damp and reflective, creating a mirror effect of the colorful lights. Many pedestrians walk about.

The Deep Learning revolution

How?

✅ **Deep Learning**

✅ **Scale! (Model, Data, Compute)**



https://openai.com/sora

Prompt: A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She walks confidently and casually. The street is damp and reflective, creating a mirror effect of the colorful lights. Many pedestrians walk about.

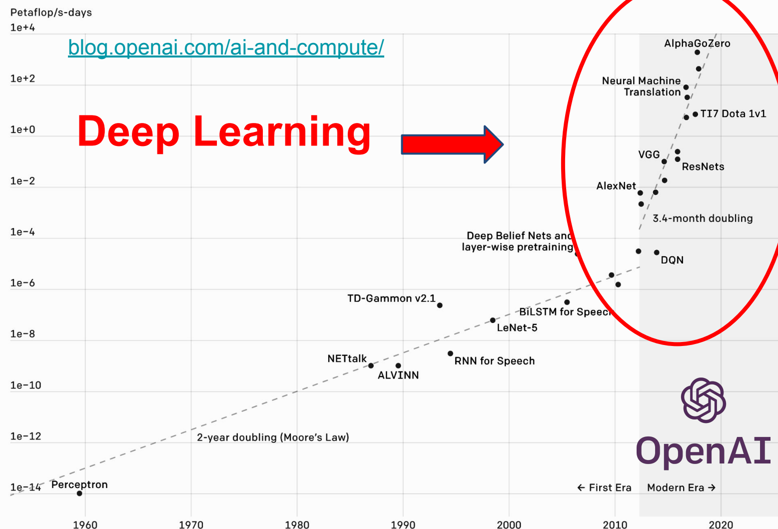
The need for HPC

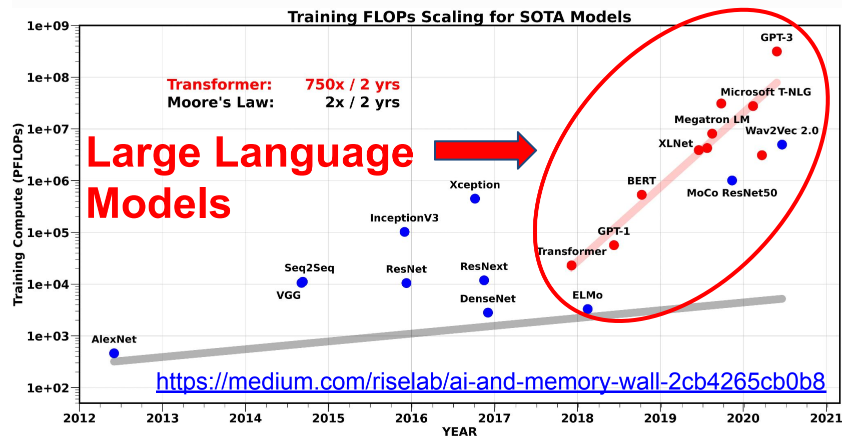
Growing computational cost of training AI models

- bigger datasets + models, more complexity

Researchers need large scale resources

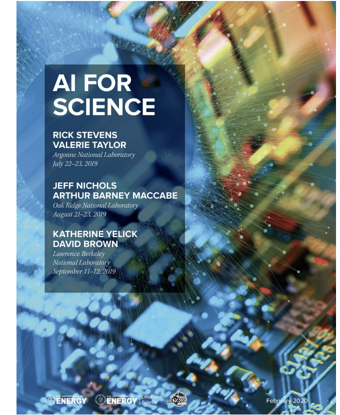
- Rapid iteration, reduce time to discovery





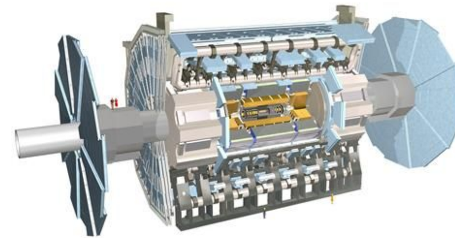
DL is transforming science

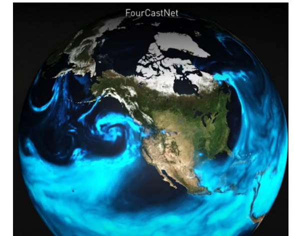
Embraced by the DOE and other funding agencies



Applied to *many* domains

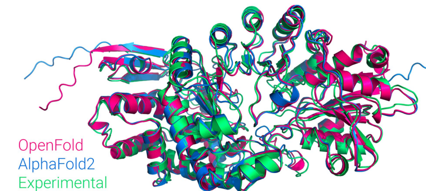
- Analyzing data better, faster
- Accelerating expensive simulations
- Control + design of complex systems



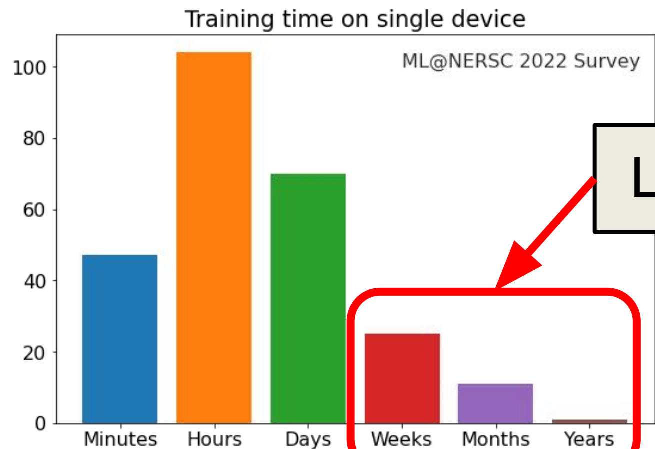


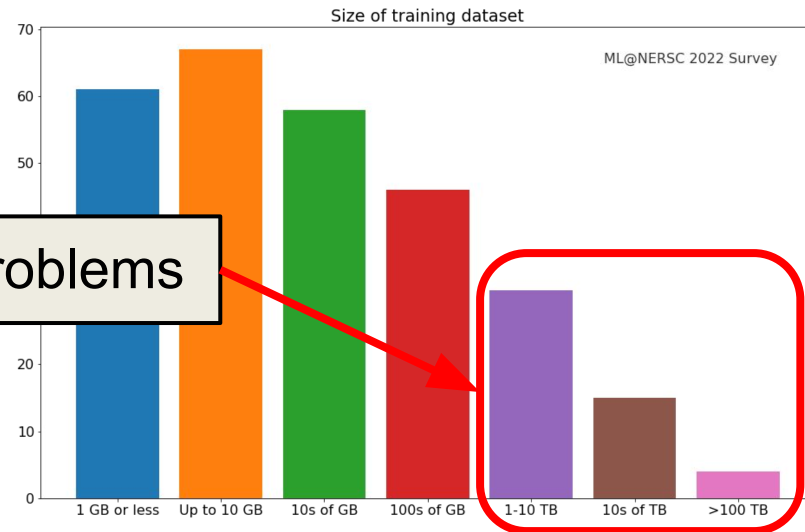
Increasingly large-scale

- Pushing limits of HPC+AI systems/tools

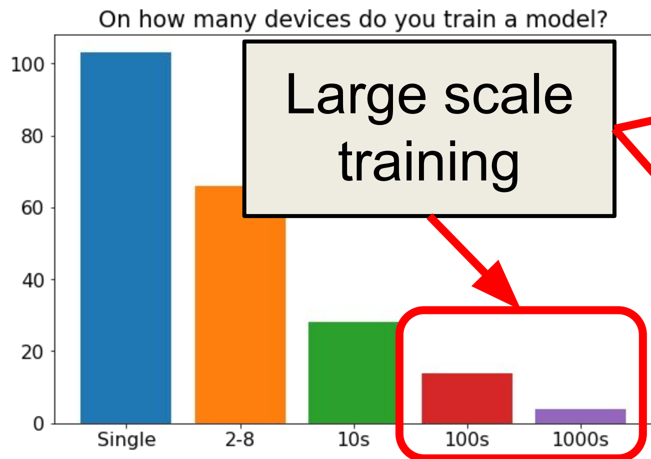


Need for AI at scale

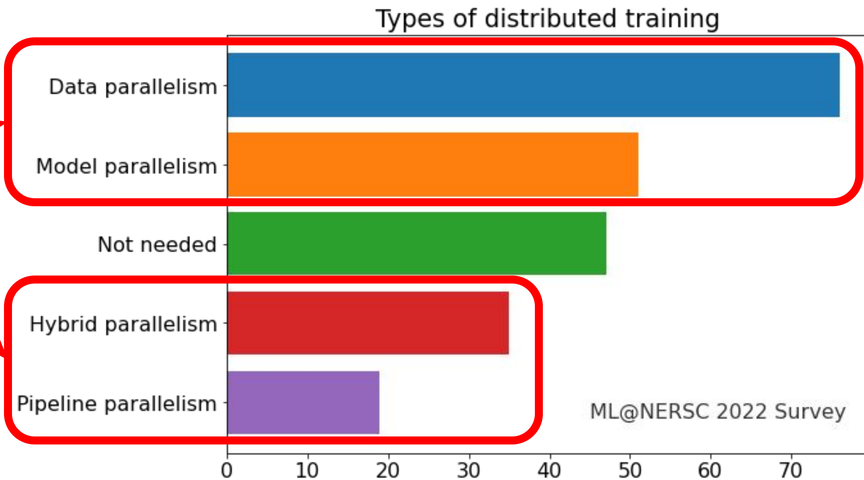


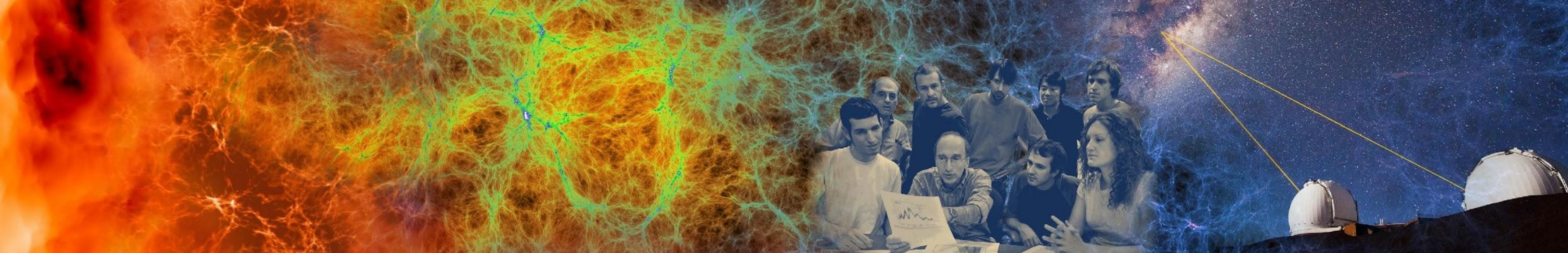


Large problems



Large scale training





DL at Scale on Perlmutter

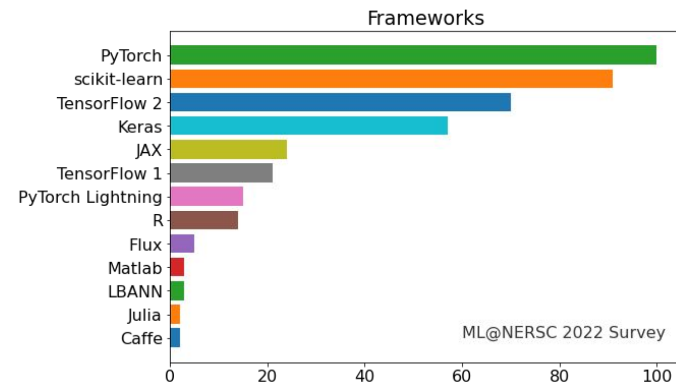
- Deep learning stack at NERSC (crash course)
- Distributed deep learning
  - Optimization & performance
  - Data parallelism
  - Model/hybrid parallelism
- Additional resources

Perlmutter deep learning software stack overview

General strategy:

- Provide functional, performant installations of the most popular frameworks and libraries
- Enable flexibility for users to customize and deploy their own solutions

Frameworks:

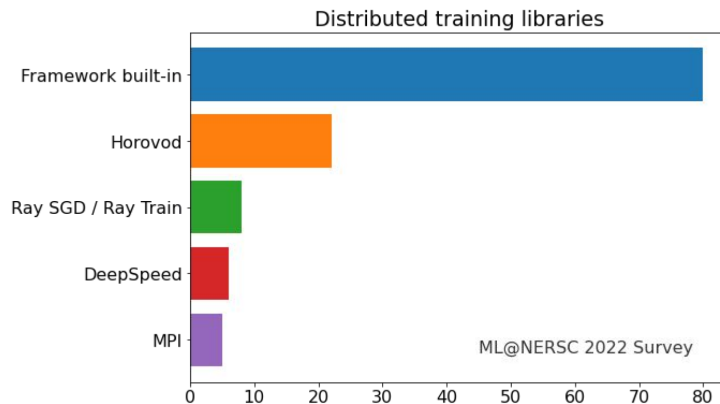


Flexibility:

- Available via pre-installed modules, custom conda/pip installations, or container builds

https://docs.nersc.gov/machinelearning/

Distributed Training Tools

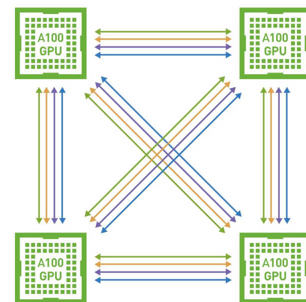


Framework built-in

- PyTorch DistributedDataParallel (DDP)
- TensorFlow Distribution Strategies

Other popular libraries

- **Horovod:** MPI+NCCL, easy to use, examples
- **Lightning:** DDP + convenient features
- **DeepSpeed:** ZeRO optimizations, 3D parallelism
- **Ray:** DDP + HPO
- **LBANN:** multi-level parallelism, ensemble learning, etc., docs

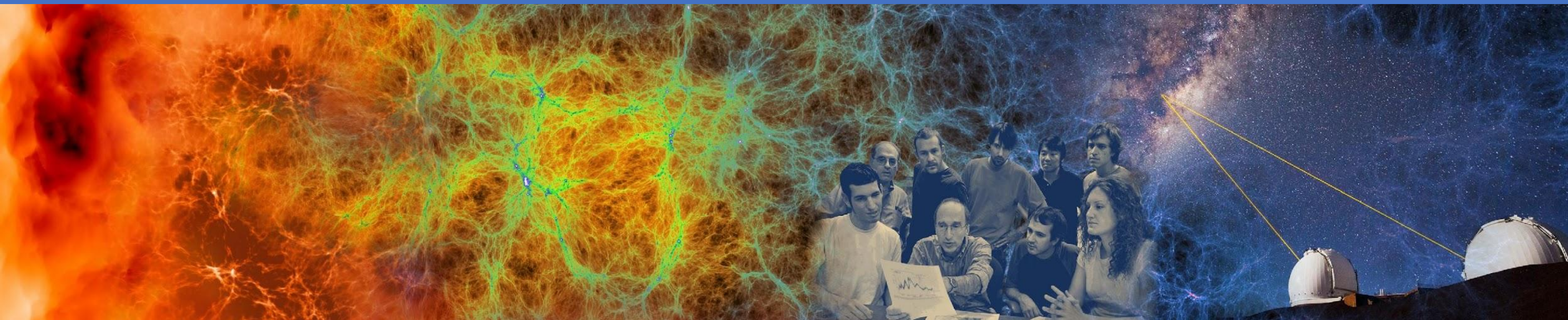


Communication backends

- NCCL is the backend of choice for GPU nodes on Perlmutter
- The NCCL OFI plugin (from AWS) enables RDMA performance on the libfabric-based Perlmutter Slingshot network (see our docs)

Distributed Deep Learning

Reference material: SC23 Deep Learning at Scale Tutorial



NERSC BERKELEY LAB U.S. DEPARTMENT OF ENERGY | Office of Science

General guidelines for distributed DL

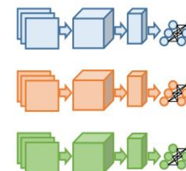
Start with an appropriate model which trains on a single CPU or GPU

Optimize the single-node / single-GPU performance

- Using performance analysis tools
- Tuning and optimizing the data pipeline
- Make effective use of the hardware (e.g. mixed precision)

Distribute the training across multiple processors

- Multi-GPU, multi-node training: data and/or model parallel
- Use best practices for large scale training and convergence
- Use best optimized libraries for communication, tune settings



Advanced parallelism

- Model/hybrid parallelism design considerations
- Implementation & analysis

Performance profiling

Using NVIDIA Nsight Systems

Profiling is an essential step in optimizing any code

Nsight Systems timeline provides a high-level view of your workload and helps you identify bottlenecks:

- I/O, data input pipeline
- Compute
- Scheduling (e.g. unexpected synchronization)

Can use NVTX ranges to annotate profiles

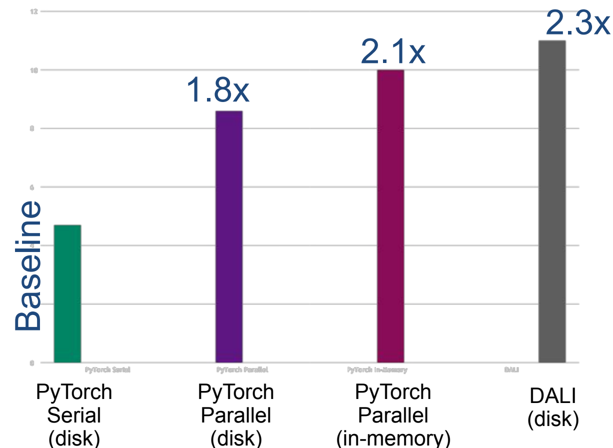
To generate a profile:

```
nsys profile -o myprofile python train.py
```

```
nsys profile -o myprofile -t cuda,nvtx python train.py
```

credit: Josh Romero, Thorsten Kurth (NVIDIA)

Optimizing GPU performance



Data loading

- Frequent cause of performance loss for users
- Parallelize your I/O
- Consider NVIDIA DALI

Mixed precision (FP32 + FP16)

- Can speed up training, leverage tensor cores, reduce memory
- Frameworks provide capabilities for automatically using FP16 where appropriate and for scaling gradients to prevent numerical underflow

JIT compilation, APEX fused operators, CUDA Graphs

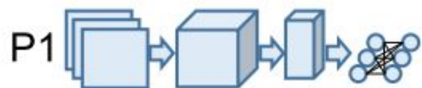
- Fuses kernels (+launches) together to increase GPU utilization

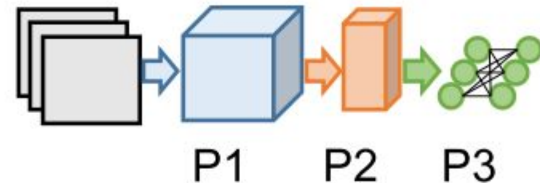
Other tricks

- Check out our tutorial for more

Full set of optimizations in tutorial => 5x speedup!

Parallel training strategies





Data Parallelism

- Distribute input samples
- Model replicated across devices
- Most common

Model Parallelism

- Distribute network structure, within or across layers
- Needed for massive models that don't fit in device memory
- Becoming more common

Parallel training strategies

Data Parallelism

- Distribute input samples
- Model replicated across devices
- Most common

The go-to for distributed DL:

✅ Conceptually simple

✅ Easy implementation

- PyTorch, TensorFlow have built-in functionality

⚠️ Some additional considerations

- Data loading at scale
- Modified hyperpameters

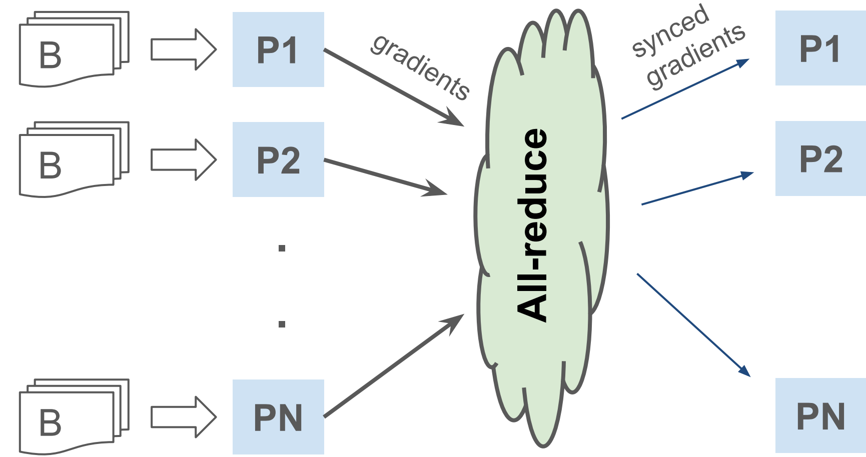
Synchronous data parallel scaling

Weak scaling (fixed local batch size)

- Global batch size grows with number of workers
- Computation grows with communication; good scalability
- Large batch sizes can negatively affect convergence

Strong scaling (fixed global batch size)

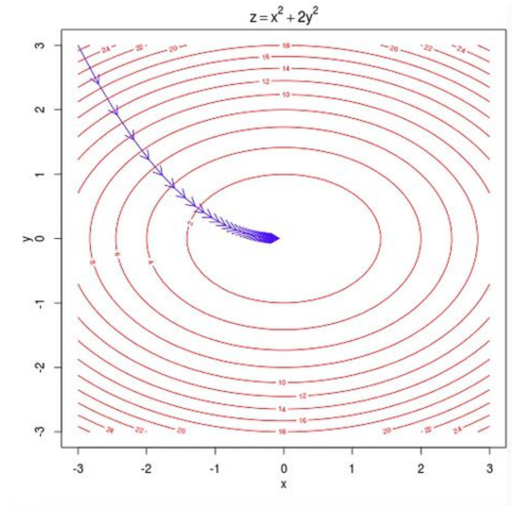
- Local batch size decreases with number of workers
- Convergence behavior unaffected
- Communication can become a bottleneck



Local batch-size = B

Global batch-size = N * B

How do we accelerate learning?



Recall batched stochastic gradient descent:

$$w_{t+1} \leftarrow w_t - \frac{\eta}{B} \sum_{i=1}^{B} \nabla L(x_i, w_t)$$

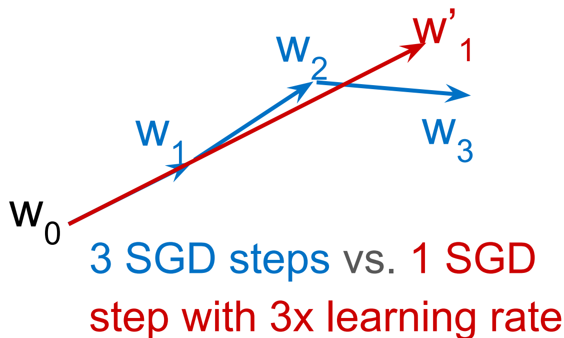
B is batch-size

η is learning rate

We can converge faster by taking fewer, bigger, faster steps

- i.e., larger batch sizes, larger learning rates, more processors
- *Not a free lunch!*

Learning rate scaling



Some rules of thumb may work for you

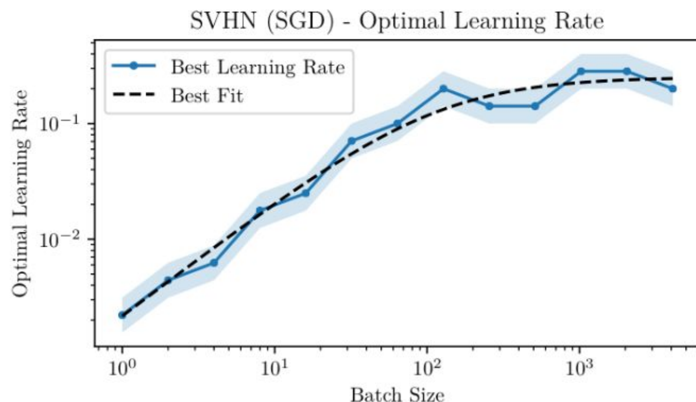
- Linear learning rate scaling:
  η → N * η
- Square-root learning rate scaling:
  η → sqrt(N) * η

Optimal learning rate can be more complex

- See OpenAI, Google studies on batchsize & learning rate co-dependence

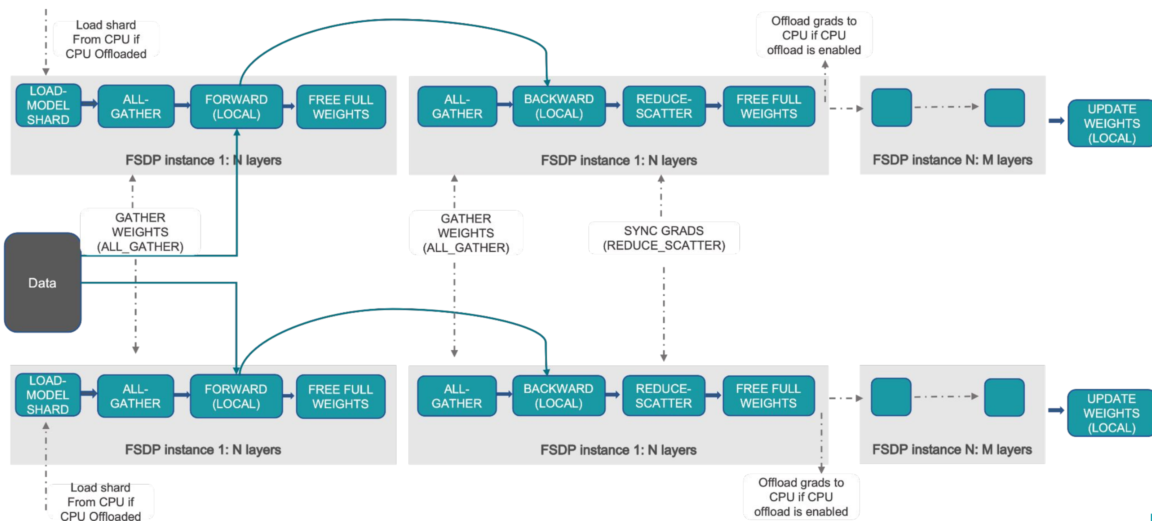
Large learning rates unstable in early training

- You may need a gradual LR "warm up"



Sharded data parallel

- Standard data parallelism fully replicates model weights and optimizer states
- We can reduce memory footprint by **sharding or offloading** these to CPU
    - Communicate parameters only when needed



https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/

Levels of sharding

- ZeRO-1: partition optimizer states
- ZeRO-2: partition gradients
- FSDP/ZeRO-3: partition weights, optionally offload to CPU

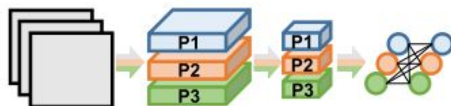
Can enable trillion parameter models without model-parallelism!

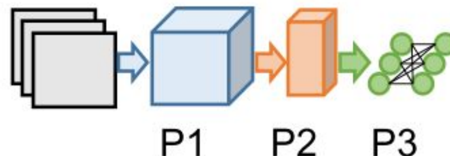
https://arxiv.org/abs/1910.02054
https://engineering.fb.com/2021/07/15/open-source/fsdp/

Beyond data parallelism

- What to do when model footprint exceeds GPU memory?
  - Data parallelism alone not enough
    - LLMs: huge models with billions of weights
    - High-res/3D/4D data: model activations dominate
  - Sharding/offloading as in ZeRO
  - Activation checkpointing

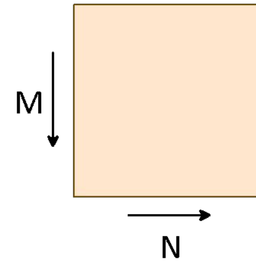
- Otherwise, **model parallelism**
  - Several tools offer various implementations

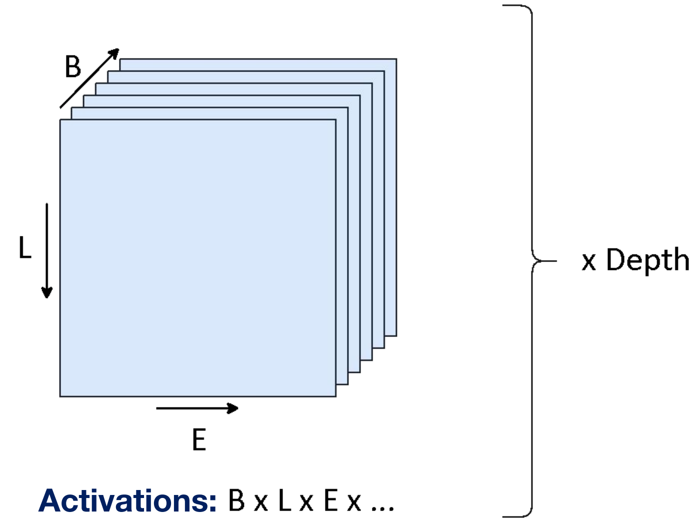




Model parallelism

- DL models are just a series of **weights** and **activations**, represented as multidimensional tensors
- Tensor dimensions determined by model architecture, input data, e.g.:
  - B: Batch size
  - D: Model depth
  - M, N: MLP weight matrices
  - L: Token sequence length (transformers)
  - E: Embedding or Feature dimension
- Picking a parallel strategy: choose which model (tensor) dimensions to partition



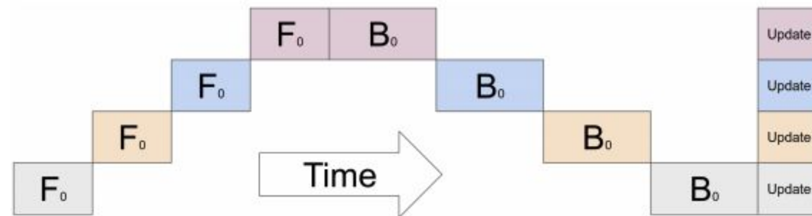


Data parallel:
Shard batch dim B

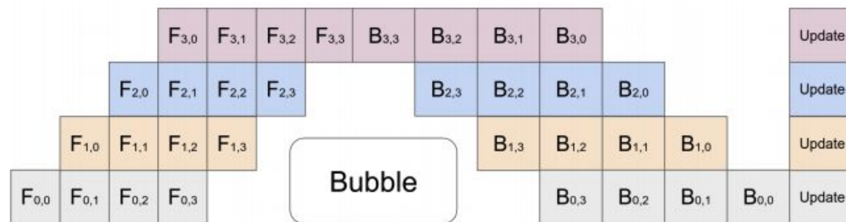
Pipeline parallel:
Shard depth D

Tensor or Operator parallel:
Shard other model dims (M,N,E)

Pipeline parallelism



1. Shard “depth” dimension across workers (different layers on different GPUs)



2. Break data batch into “microbatch” and overlap computation

Considerations:

- Idle bubbles still impact overall utilization
- Can be more straightforward than other model parallelism

```
net = nn.Sequential(
    nn.Linear(in_features, hidden_dim),
    nn.ReLU(inplace=True),
    nn.Linear(hidden_dim, out_features)
)
from deepspeed.pipe import PipelineModule
net = PipelineModule(layers=net, num_stages=2)
```

https://www.deepspeed.ai/tutorials/pipeline/

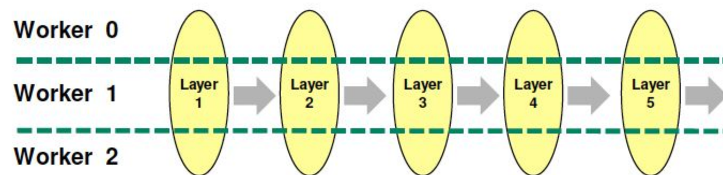
Tensor/Operator parallelism

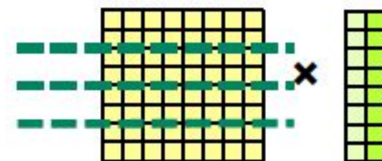
- Shard other tensor dimensions across workers

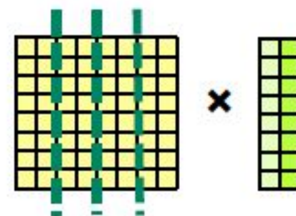
- Full flexibility, choices are model/data-dependent, e.g.:
  - Transformers – parallelize MLP matrix multiplies row-wise or col-wise
  - CNNs – spatial parallelism (domain decomposition)

- Communication in forward/backward pass depends on what is sharded and how

- Addresses some of the challenges of pipelining (idle slots, load imbalance); more involved to implement
  - Custom forward/backward pass implementations for different communication patterns
  - Ref. SC23 material for advanced use-cases

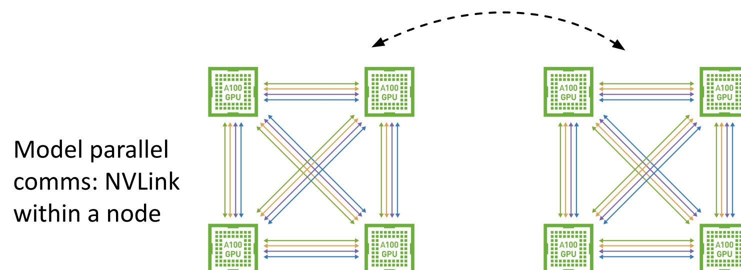




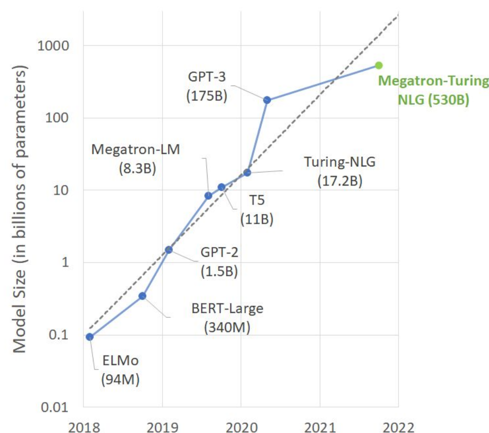


Hybrid parallelism

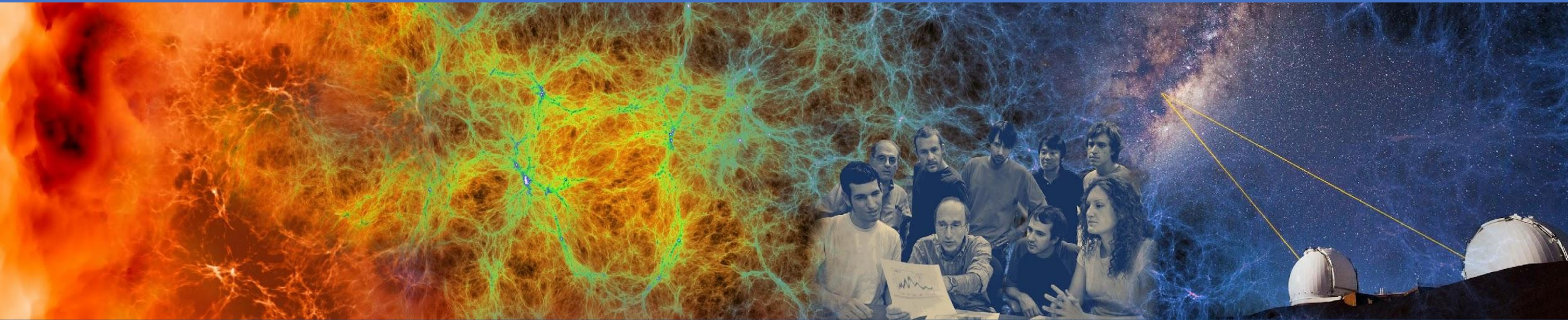
- Data + Model parallel at the same time!
  - Need multiple communicator groups
  - Prioritize high-bandiwdth (NVLink) for ops that do the most frequent/largest communication



- Used by most SOTA extreme-scale DL models, e.g NVIDIA MegatronLM implementation of GPT3:
  - 8-way tensor parallelism on node
  - 16-way pipeline parallelism
  - Data parallelism up to thousands of GPUs

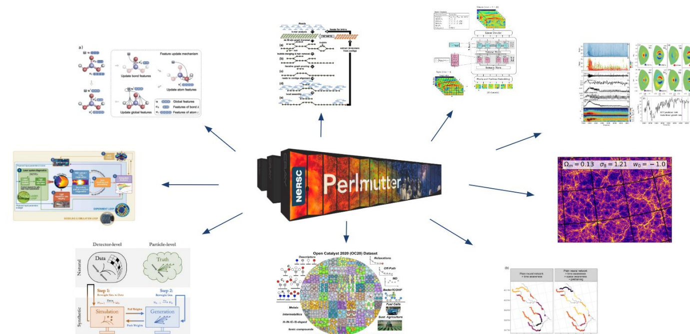


Outreach & additional resources



NERSC BERKELEY LAB U.S. DEPARTMENT OF ENERGY | Office of Science

Outreach & additional resources



NESAP engagements

- A major way of engaging on advanced AI use-cases
- Science team partners with NERSC staff
- Forward-looking, e.g. towards N10 workflows
- CFP likely at the end of FY24

The Deep Learning at Scale Tutorial

- Jointly organized with NVIDIA (+ previously Cray, ORNL)
- Presented at SC18-23, ECP Annual 2019, ISC19
- Detailed lectures + hands-on material
- Runs on Perlmutter!

NVIDIA AI for Science Bootcamp

- More methods-focused, but relevant to scientific computing
- 2022 event, 2023 event

Questions?
Collaboration? Want help?

Peter Harrington
pharrington@lbl.gov

Deep-learning@NERSC:
https://docs.nersc.gov/machinelearning/

Join the NERSC Users Slack