GNN Scaling – next steps

Steve Farrell
NERSC, LBNL

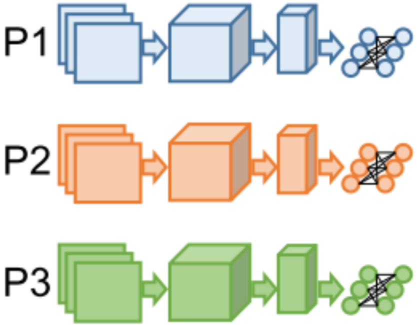
Exa.TrX F2F, 2020-04-07

Why should we use distributed training?

- We said we'd use HPC + distributed training in our proposal ;)
- It allows to more quickly train large models on large, complex datasets
- We have large, complex graphs; a lot of potential intra-event parallelism
- We can have large simulated datasets in HEP
- Publishing research on large scale GNN training will be valuable to the community

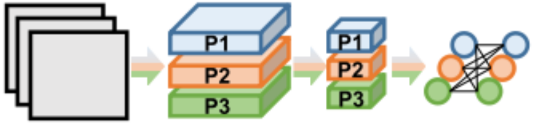
NERSC

Parallelism strategies



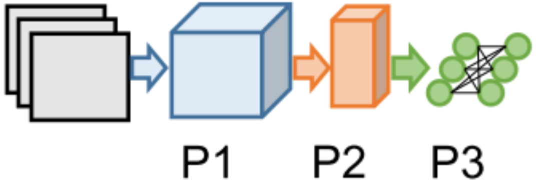
Data Parallelism

Distribute input samples.



Model Parallelism

Distribute network structure (layers).



Layer Pipelining

Partition by layer.

Fig. credit: Ben-Nun and Hoefler arXiv:1802.09941

Data parallelism, synchronous Updates

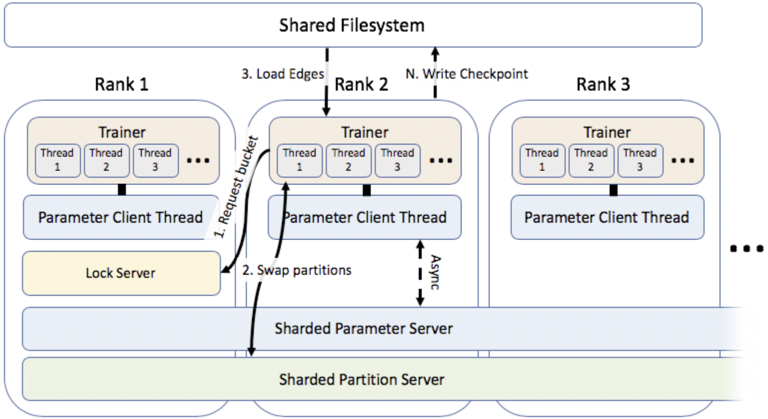
Gradients are computed locally and summed across nodes. Updates are propagated to all nodes

- stable convergence
- scaling is not optimal because all nodes have to wait for reduction to complete
- global (effective) batch size grows with number of nodes

Synchronous SGD, decentralized

Parallelizing Graph Neural Networks

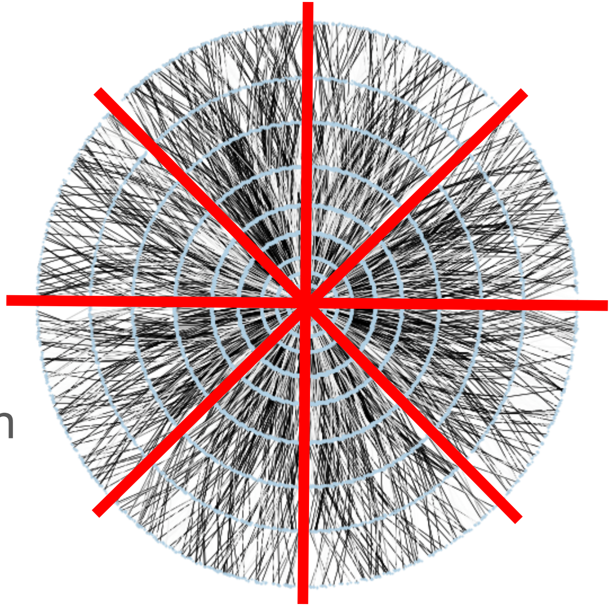
- Most deep learning literature on scaling training is on *computer vision applications*
- Graph deep learning is much newer, so methods for scaling are much less established
- Also, graph deep learning is a *diverse* set of applications, not all of which are applicable to us
  - E.g., FaceBook has (probably) the largest social network graph, but it's essentially just one enormous graph



PyTorch-BigGraph: https://arxiv.org/abs/1903.12287
"scale to graphs with billions of nodes and trillions of edges"

Our GNN parallelism

- Naïve domain parallelism
  - Split into sectors, train on them independently
  - You don't need the whole detector context to find tracks in a region
  - (Minor) technical challenges in doing inference on a whole event
  - Small batch sizes are good for generalization
- Proper domain parallelism
  - Break graph into sectors/partitions, but handle the boundaries with communication
  - Definitely more difficult to implement



Other approaches also under consideration

HPC scaling work with Cray BDC

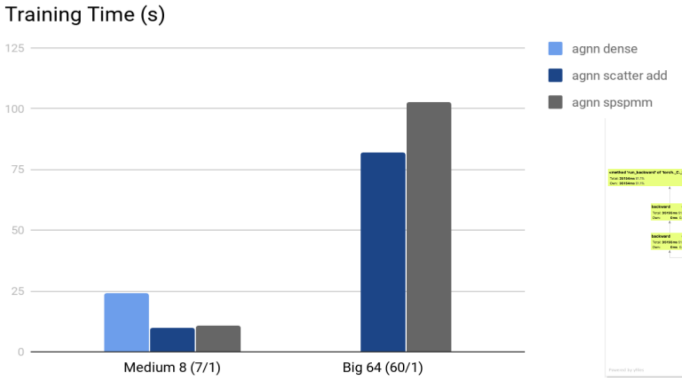
- **Through the Cray Big Data Center collaboration, we’re engaging with folks from Cray and LBNL’s CRD to push on HPC scaling of GNNs (for tracking)**
  - Strong interest in scaling GNNs in PyTorch
  - The basic plan is to do large scale training of GNNs in a larger Population Based Training run
- **Computational challenges**
  - Graphs with sparse connectivity => need sparse op support
  - Variable sized graphs => need to handle load imbalance at scale
  - Large scale training of GNNs => not much experience/intuition
- **Using my PyTorch implementation of message-passing and “attention” networks here:** https://github.com/sparticlesteve/heptrkx-gnn-tracking

NERSC

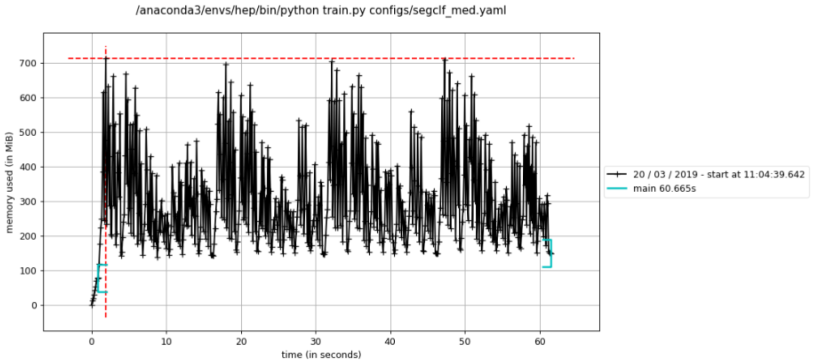
Single node performance [Saliya Ekanayake, LBNL]

- Compared speed and memory of dense and multiple sparse representations

	Training Time (s)		
Dataset N (tr/val)	**agnn dense**	**agnn scatter add**	**agnn spspmm**
Med 8 (7/1)	24.0416	9.63175	10.6389
Big 64 (60/4)		81.9128	102.74

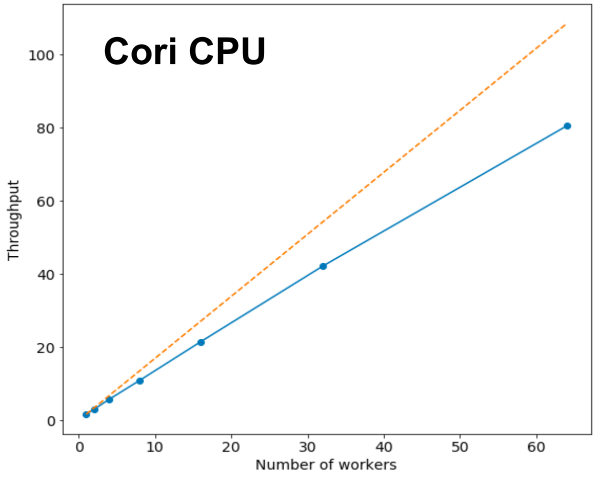


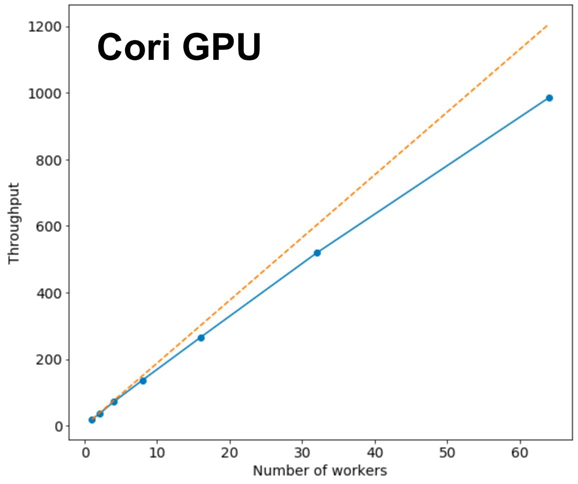
- **PyTorch-Geometric was the best of what we tested in terms of speed and memory**

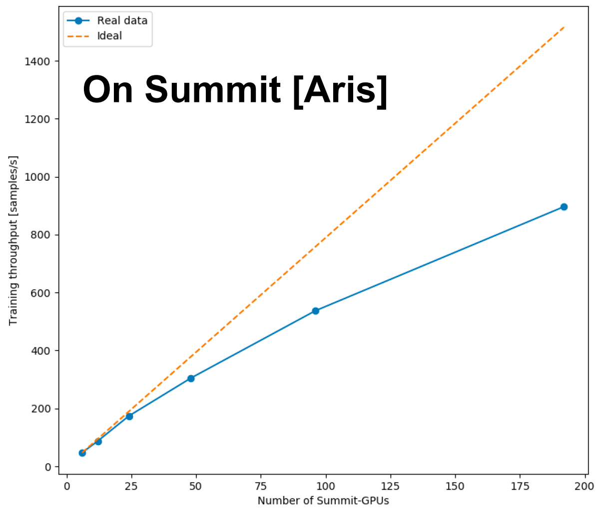


Scaling

- Distributed training scaling on OLCF and NERSC machines
- Scaling efficiency is not bad, actually, considering that it's expected to be adversely affected by load imbalance

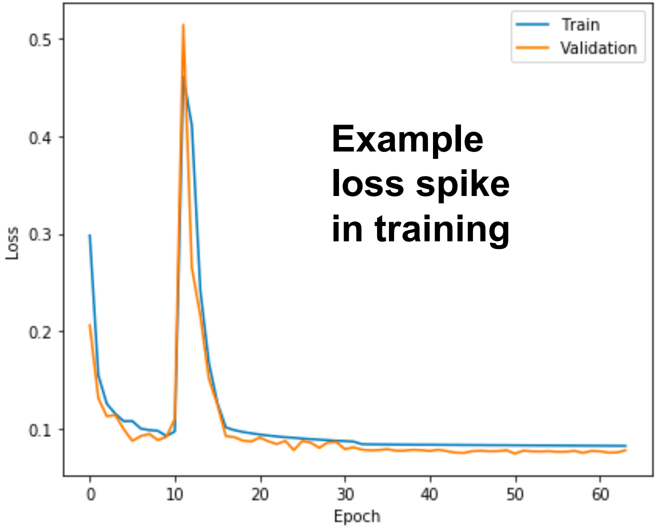






Training instabilities

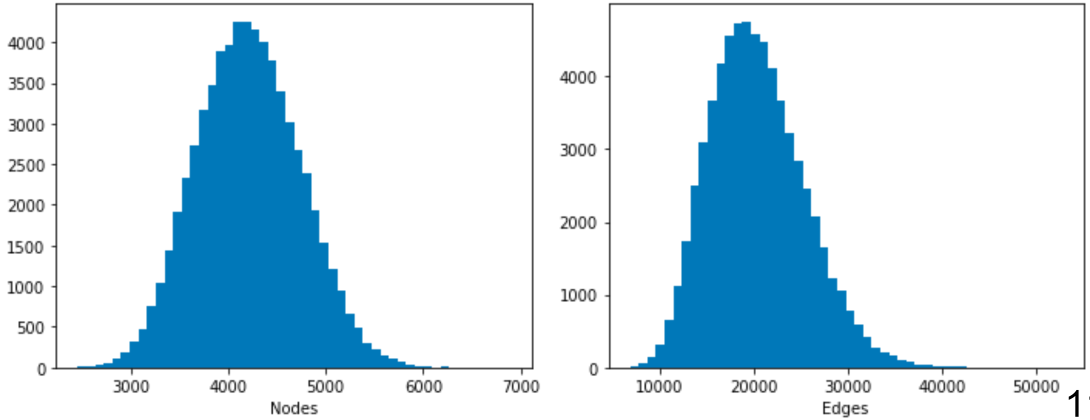
- Occasionally suffering from spikey/unstable behavior in the training loss when training distributed
- I've spent a bit of time digging into this
  - Tracking gradient norms, weight norms
  - Reducing graph size variance
  - Gets worse with larger models
  - Improved somewhat with layer norm, weight decay
- It was not fully solved, but I expect it's related to the interactions between, class weights (real vs. fake edges), variable sample sizes and purities, optimizer momentum, and gradient reductions as averages of averages.
- Other things expected to help
  - Stabilize training with auxiliary targets (e.g. predicting p_T)
  - Balance data sampling



Balanced data sampling

- Distribution of graph sizes leads to load imbalance
- Solution in development, inspired by the work done in Etalumis project: https://arxiv.org/abs/1907.03382
  - Bin dataset into buckets of similar "size"
  - Sample batches from these buckets
- There can be effects on convergence, which we'll need to study
  - It worked for Etalumis, though

Graph size variability



Outlook and next steps

- This work stalled because of lack of time, but is now being picked back up
- Our original plan was to use Cori KNL for a large scale study (and submit to something like SC, IPDPS), but this has been abandoned
  - KNL speed was factor ~8 slower than Haswell, would require considerable effort with Intel to improve; decided not worth it
  - Haswell system could still be useful, though it is in high demand nowadays
- Current plan is to target a smaller system with GPUs (e.g. Cori-GPU) to wrap up the work with Cray, and submit to a workshop
- After that, there are more fun things to do
  - Push further on scaling, run on Summit and upcoming Perlmutter
  - Smarter graph partitioning/parallelization
  - Scale the newer methods explored by Nick and Daniel (and others)

NERSC