Exascale Deep Learning for Climate Analytics

Ron Davies

AU 21
• Introduction and motivation
• System innovations
• Deep learning innovations
• Performance
• Critiques
Is the Earth's climate changing in consequential ways?
• 50% of rainfall in California comes through atmospheric rivers
  • What could happen to these?
• $200B in damages in 2017 due to tropical storms
  • Will landfall and severity increase and where?
• High fidelity weather simulations produce 10s of TBs of output requiring automated analysis
• Deep learning approaches have been successful at detecting features of interest
• This paper pushes for pixel-level segmentation masks of the weather patterns
Segmentation architectures:
- DeepLabv3+
- Tiramisu (extended ResNet)

Data and ground truth:
- 63K high-resolution samples (3.5 TB)
- 80/10/10 train/test/validation
- Composition:
  - 98.2% background
  - 1.7% atmospheric rivers
  - <0.1% tropical cyclones
• Introduction and motivation
• System innovations
• Deep learning innovations
• Performance
• Critiques
They present 3 system innovations:

• Fast parallel data staging
• Optimized data ingestion
• Hierarchical all-reduce
Fast parallel data staging: (Summit)

- **Need:** 5.23 TB/s (>2x GPFS performance)
- **Try:** Independently select a set of samples to form necessary batches at each rank
  - 250 images per GPU and 1500 per node

What might happen if you do this when using 4,560 nodes (27,360 GPUs)?
Fast parallel data staging: (Summit)

- **Result:** 20 minutes of staging and made the file system unusable for others

- **Solution:**
  - Disjoint sets of the data are read by each rank (multithreaded)
  - Use InfiniBand and MPI messages to distribute copies of data
  - Data is distributed to 1024 nodes in 3 minutes
Optimized data ingestion:

- **Need**: Data loaded as tensors before requested because this is a CPU process, and the GPU will remain idle
- **Try**: TensorFlow prefetching options
- **Result**: The queue was not always full due to HDF5 library limits
- **Solution**: Spawn multiple parallel processes which keep up even for FP16 precision
Hierarchical all-reduce:

- Distributed training handled by Horovod
- Horovod (~2018) efficiency held back by 2 factors:
  - Centralized reordering of all-reduce operations (a Horovod feature) maintains consistency at the cost of high message passing totals for large jobs
  - Utilization of either MPI_Allreduce OR NCCL
Hierarchical all-reduce:

- Create a nested tree structure with radix r
- Each node waits to send a readiness message to its parent until after they receive those from all their children and finish their own computation
- The rank 0 node at root of the tree uses base Horovod
- Root coordinates with r+1 children
- Reduces the number of messages at rank 0 from millions to thousands of messages per second.
Hierarchical all-reduce:

- Horovod uses either MPI-Allreduce or NCCL
- Authors use a hybrid method
- Within each node, all 6 ranks use NCCL to reduce
- 4 of the 6 then use MPI_Allreduce with $\frac{1}{4}$ of the data to communicate with all other nodes

This takes advantage of the faster NVLink speeds between GPUs in a node
• Introduction and motivation
• System innovations
• Deep learning innovations
• Performance
• Critiques
Deep learning innovations:

- **Weighted loss:** inverse square root of class frequency
- **LARC:** layer-wise adaptive rate control keeps the magnitude of weight updates small relative to the norm of layer weights
Deep learning innovations:

- **Multi-channel segmentation:** they used 16 channels on Summit though they believe the optimal number may be fewer

- **Gradient lag:** weight updates use the gradients computed in the previous step

- **Architecture modifications:**
  - Use a growth rate of 32 instead of 16 for Tiramisu
  - Use the full resolution decoder instead of ¼ resolution in DeepLabv3+
• Introduction and motivation
• System innovations
• Deep learning innovations
• Performance
• Critiques
Measures and challenges:

• Profiling the operations to determine FLOP count per step

• Sampling images to normalize the FLOP count per step before computing statistics on a time series of steps

• TensorFlow dynamically picks the algorithm of choice for performance
Weak scaling results:

- Dotted line represents ideal scaling
- Both levels of precision appear close to the scaling results
- Gradient lag makes a big difference
- 90.7% parallel efficiency
Climate Science:
- 59% | Tiramisu network
- 73% | DeepLabv3+
- Qualitatively, it looks visually pleasing
- Good approximation of exact boundaries
Concise findings:

• Modifications to the deep learning model (lag, weighted loss, etc.,) accelerated training on each node
• Modifications to data loading, preparing, and all-reduce operations kept GPU idle time minimal
• These methods can be used to apply deep learning to a wide variety of scientific disciplines
• This is the first exascale-class deep learning application
Critique

Gradient lag played one of the biggest roles in pushing the sustained performance from <600PF/s to 999PF/s, but it was given minimal discussion space

Kurth et al., Exascale Deep Learning for Climate Analytics (SC '18 Gordon Bell Winner)