Machine Learning/Deep Learning on Summit

Junqi Yin
Advanced Data and Workflow
Oak Ridge Leadership Computing Facility
Outline

• Overview
• ML/DL software stack on Summit
• Distributed deployment considerations
• Performance baselines
  – ML: PCA, Kmeans, SVM, XGBoost
  – DL: image (ResNet), text (BERT)
• Scaling practices
  – Compute, I/O, Communication
• Conclusion
ML/DL applications on Summit overview

- ML/DL has entered exascale computing
  - (1) “Exascale Deep Learning for Climate Analytics”
  - (2) “Exascale Deep Learning to Accelerate Cancer Research”
  - (3) “Exascale Deep Learning for Scientific Inverse Problems”

<table>
<thead>
<tr>
<th>Application</th>
<th>Network</th>
<th>Sustained Performance (ExaFlops)</th>
<th>Peak Performance (ExaFlops)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Climate</td>
<td>DeepLabV3+</td>
<td>0.999</td>
<td>1.13</td>
</tr>
<tr>
<td>(2) Medical</td>
<td>MENNDL</td>
<td>1.3</td>
<td>n/a</td>
</tr>
<tr>
<td>(3) Materials</td>
<td>Tiramisu variant</td>
<td>1.5</td>
<td>2.1</td>
</tr>
</tbody>
</table>
ML/DL applications on Summit overview

- ML/DL by domains and methods (growing list)

<table>
<thead>
<tr>
<th>Domain\Method</th>
<th>Supervised Learning</th>
<th>Unsupervised Learning</th>
<th>Reinforcement Learning</th>
<th>Hyperparameter Search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random Forest</td>
<td>MLP</td>
<td>CNN</td>
<td>RNN (LSTM)</td>
</tr>
<tr>
<td>Biophysics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chemistry</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Climate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Computer Science</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fusion</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Life Sciences</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Material Sciences</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medical Sciences</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nuclear Physics</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turbulence</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Particle Physics</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
ML/DL stack on Summit

Machine learning

Deep learning

Frameworks

Plugins and Tools

Visualization & Workflow: ParaView, Visit, SIGHT, Pegasus, Radical Toolkit, …
Distributed deployment considerations

- Framework consideration
  - TensorFlow vs PyTorch
  - NCCL vs MPI backend
Distributed deployment considerations

- Native vs Container
  - Impact of loading shared libs: manageable on GPFS
  - Runtime performance: comparable
Performance baselines: PCA

- Tall skinny input
- Single node
  - Up to 10 GB/s (32 GB input)
  - Single GPU + UVM < Multi-GPU

CuML PCA on single Summit node

Processing speed (GB/s)

- Strong vs weak scaling
- Rapids is evolving
Performance baselines: Kmeans

- Tall skinny input
- Single node: up to 5.4 GB/s (24 GB input)
- Strong vs weak scaling

**cuML Kmeans on single Summit node**

- # of columns=250

**cuML Kmeans scaling**

- # of columns=250

Graphs show the processing speed (GB/s) vs input size (GB) and the number of Summit nodes. The graphs compare performances for single GPU (V100), Multi GPUs (6 V100), and pbdR (2 P9) for both strong and weak scaling scenarios.
Performance baselines: SVM, XGBoost

- SnapML SVM better strong scaling
- Rapids cuDF + Dask + XGBoost

GTC20 Slides
Performance baselines: ResNet50 on ImageNet

TF_CNN_Benchmark on Summit: ResNet50
batch-size = 256 per GPU

Images/second

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Images/second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5320</td>
</tr>
<tr>
<td>8</td>
<td>41746</td>
</tr>
<tr>
<td>64</td>
<td>321492</td>
</tr>
<tr>
<td>128</td>
<td>631664</td>
</tr>
<tr>
<td>256</td>
<td>1227241</td>
</tr>
<tr>
<td>512</td>
<td>2408388</td>
</tr>
<tr>
<td>1024</td>
<td>4731623</td>
</tr>
</tbody>
</table>

Top-1 Val accuracy

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Mini-batch size</th>
<th>Top-1 Val accuracy</th>
<th>Training time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>12288</td>
<td>0.750</td>
<td>27</td>
</tr>
<tr>
<td>32</td>
<td>12288</td>
<td>0.766</td>
<td>17</td>
</tr>
<tr>
<td>64</td>
<td>15360</td>
<td>0.763</td>
<td>12</td>
</tr>
</tbody>
</table>

TF.distribute & Horovod + LARS

code: TensorFlow distributed example
Performance baselines: BERT on Wikipedia

- Throughput: 63% (phase1) 83% (phase2)
- Time-to-Solution: ~64min on 1536 V100 vs ~76min on 1024 TPUv3 (arXiv:1904.00962)

**code: PyTorch BERT example**  
Apex DDP + LAMB
Scaling practices

1. Compute:
   - Tune on single node with synthetic data

2. I/O
   - Tune on NVMe with input pipelining

3. Communication
   - Tune at scale with comm. libs

https://sc19.supercomputing.org/presentation/?id=ws_dls120&sess=sess101

Tune single node compute with synthetic data
Tune single node I/O with input format and framework
Tune communication library with application at scale
Always checkpointing

- It is relatively straightforward and cheap to checkpoint in DL.
- for data parallel, it is essentially the same as for single GPU.

```python
# Save checkpoint
if hvd.rank() == 0:
    state = {
        'model': model.state_dict(),
        'optimizer': optimizer.state_dict(),
    }
torch.save(state, filepath)

# Load checkpoint
if hvd.rank() == 0:
    checkpoint = torch.load(filepath)
    model.load_state_dict(checkpoint['model'])
    optimizer.load_state_dict(checkpoint['optimizer'])

# Horovod: broadcast parameters & optimizer state.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
hvd.broadcast_optimizer_state(optimizer, root_rank=0)
```
Scaling considerations: Compute

- Use Tensor Cores (2 ~ 4x) 15 TFLOPS(FP32) vs 120 TFLOPS(FP16)
  - PyTorch: NVIDIA Apex plugin https://github.com/NVIDIA/apex
    
    ```
    # Added after model and optimizer construction
    model, optimizer = amp.initialize(model, optimizer, flags...)
    # loss.backward() changed to:
    with amp.scale_loss(loss, optimizer) as scaled_loss:
      scaled_loss.backward()
    ```

  - TensorFlow:
    
    ```
    #Enable TF-AMP graph rewrite:
    os.environ["TF_ENABLE_AUTO_MIXED_PRECISION_GRAPH_REWRITE"] = "1"
    #Enable Automated Mixed Precision:
    os.environ["TF_ENABLE_AUTO_MIXED_PRECISION"] = '1'
    ```

  - Verify on Tensorcore:
    
    ```
    #Turn off Tensorcore:
    os.environ["TF_DISABLE_CUDNN_TENSOR_OP_MATH"] = "0"
    #nvprof: tensor_precision_fu_utilization to show TC utilization
    ```
Scaling considerations: Compute

• Use XLA (~ 1.5x for ResNet50)

```python
#Enable XLA:
session_config.graph_options.optimizer_options.globaljit_level =
tf.OptimizerOptions.ON_1
```

• Tune cuDNN algorithms (e.g. 7 implementations for conv)

```bash
#TensorFlow
os.environ['TF_CUDNN_USE_AUTOTUNE'] = '1'
#PyTorch
torch.backends.cudnn.benchmark = True
```

• Know your kernels (optimal scheduling policy)
Scaling considerations: Compute

• Benchmark kernels
  – CNN:
    kernel size, # of kernels, etc.
  – RNN:
    batch size, timesteps, etc.
Scaling considerations: I/O

- Use node local NVMe

<table>
<thead>
<tr>
<th>Device \ Bandwidth</th>
<th>Full system run</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPUs processing</td>
<td>$3 \times 224 \times 224 \times 4 \times 1200 \times 6 \times 4608 / 10^{12} \approx 20 \text{ TB/s}$</td>
</tr>
<tr>
<td>GPFS reading</td>
<td>2.5 TB/s</td>
</tr>
<tr>
<td>NVMe reading</td>
<td>6 GB/s * 4608 \sim 27 \text{ TB/s}</td>
</tr>
</tbody>
</table>

- Use LMDB input format

I/O read throughput for 6 ranks on a Summit node
Uncompressed ImageNet data, batch=256 per GPU

TFRecord  \quad HDF5  \quad LMDB

<table>
<thead>
<tr>
<th>SMT = 1</th>
<th>SMT = 2</th>
<th>SMT = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scaling considerations: I/O

• CPU affinity settings (for pre-processing OPs on CPU)
  – Correct binding (numactl + OMP_PLACE):

    # for SMT4
    case $((($PMIX_RANK%6)) in
      [0])
      export PAMI_IBV_DEVICE_NAME=mlx5_0:1
      export OMP_PLACES={0:28}
      numactl --physcpubind=0-27 --membind=0 $APP
    ;;
    [1])
    export PAMI_IBV_DEVICE_NAME=mlx5_1:1
    export OMP_PLACES={28:28}
    numactl --physcpubind=28-55 --membind=0 $APP
    ...;

• Use pre-processing pipeline [https://www.tensorflow.org/guide/data_performance] for SMT4: staging, prefetch, parallel_interleave, etc
Scaling considerations: Communication

- Communication libraries
  - Low level: NCCL, MPI
  - High level: Hovorod, Apex

- Framework support

<table>
<thead>
<tr>
<th>Framework \ Distribution</th>
<th>Single node</th>
<th>Multi node</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>MirroredStrategy</td>
<td>MultiWorkerMirroredStrategy</td>
</tr>
<tr>
<td>PyTorch</td>
<td>DataParallel</td>
<td>DistributedDataParallel</td>
</tr>
</tbody>
</table>
Scaling considerations: Communication

- Use Horovod with NCCL backend

**TensorFlow**

```
0 200 400 600 800 1000
1 2 4 8 16 32 64 128

# of compute nodes
```

**PyTorch**

```
0 50 100 150 200 250 300 350 400
16 32 64 128 256

# Summit nodes
```
Scaling considerations: Communication

• Tune Horovod parameters
  – Key knobs: HOROVOD_CYCLE_TIME, HOROVOD_FUSION_THRESHOLD

#Horovod autotuner
export HOROVOD_AUTOTUNE=1
export HOROVOD_HIERARCHICAL_ALLGATHER=0
export HOROVOD_HIERARCHICAL_ALLREDUCE=0
export NCCL_DEBUG_SUBSYS=COLL
Full Summit Scaling: A Material Science App

- Sustained performance: 1.5 EFLOPS (FP16)
- 93% scaling efficiency at 4600 nodes

Peak Performance: **2.1 EFLOPS (FP16)**

Conclusion

- Summit is ideal for ML/DL applications
- ImageNet training with ResNet50 can achieve 87% scaling efficiency up to 1024 nodes
- BERT per-training on Wikipedia corpus in about 1 hour on 256 nodes
- Multi-node multi-GPU PCA, Kmeans, SVM, XGBoost etc with Rapids, SnapML, and pbdR

For more information:
- Scaling up DL on Summit webinar: slides recording
- https://code.ornl.gov/olcf-analytics/summit/distributed-deep-learning-examples
- https://github.com/benjha/nvrapids_olcf
- https://code.ornl.gov/olcf-analytics/summit/rapids

Thank You!
More resources

• More examples (most run on Summit with minimum changes)
  – Horovod examples: https://github.com/horovod/horovod/tree/master/examples

• Official documentations
  – TensorFlow distributed training: https://www.tensorflow.org/guide/distributed_training
  – PyTorch data parallel: https://pytorch.org/tutorials/intermediate/ddp_tutorial.html