Early experiences with Machine Learning and Deep Learning on Summit/Summit-Dev

Junqi Yin
Advanced Data and Workflows Group
Outline

• ML/DL software stack on Summit
• CORAL2 benchmark
  – Data Science benchmark
    • Big Data Analytics Suite
    • Deep Learning Suite
• ML/DL performance model: Summit-Dev to Summit
• Scaling DL
  – Resnet50 on ImageNet
  – Lessons learned from exa-scale DL on Summit
• Discussion: ML vs DL use cases
ML/DL software stack on Summit (current plan and subject to change)

- Native installation
- IBM PowerAI container
- Custom container with Singularity (in planning)

<table>
<thead>
<tr>
<th>Framework</th>
<th>Native</th>
<th>PowerAI Container</th>
<th>Custom Container</th>
<th>Python Wheels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensorflow</td>
<td>1.12</td>
<td>1.10, 1.8</td>
<td>1.9</td>
<td>tensorflow-1.12.0-cp36-cp36m-linux_ppc64le.whl</td>
</tr>
<tr>
<td>Pytorch</td>
<td>1.0rc1</td>
<td>0.4.1</td>
<td>0.4.1</td>
<td>torch-1.0.0a0+ff608a9-cp36-cp36m-linux_ppc64le.whl</td>
</tr>
<tr>
<td>R/PbdR</td>
<td>1.1</td>
<td></td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>SnapML</td>
<td>1.0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/gpfs/wolf/stf011/world-shared
### ML/DL software stack

<table>
<thead>
<tr>
<th>Framework/Version</th>
<th>Native</th>
<th>PowerAI Container</th>
<th>Custom Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensorflow 1.12.0</td>
<td>1.10</td>
<td>1.10</td>
<td>1.8</td>
</tr>
<tr>
<td>Pytorch 1.0</td>
<td>0.41</td>
<td>0.41</td>
<td>0.41</td>
</tr>
<tr>
<td>SnipML 1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Wheels**
- CUDA 9.2.148/CUDNN 7.4.1/NCCL 2.3.7
- Tensorflow 1.12.0-qpl6-40p36m-linux_pgf44a.wrl
- Pytorch 1.0.0a0-f5609a-4q50-cp56m-linux_pgf64a.wrl

**Documentation**
- PowerAI on Summit
- Tutorial
  - Keras/Tensorflow on Summit

[GitHub Repository 1](https://code.ornl.gov/jqyin/mldl-hpc)
[GitHub Repository 2](https://code.ornl.gov/summit/mldl-stack)
**CORAL-2 Data Sciences Benchmarks**

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Big Data Analytic Suite</td>
<td>PCA, K-Means, and SVM (based on pbdR)</td>
</tr>
<tr>
<td>Deep Learning Suite</td>
<td>CANDLE, CNN, RNN, and ResNet-50 (distributed)</td>
</tr>
</tbody>
</table>

Deep Learning Codes (CNN; ResNet50; ..) excel here with NVM and GPUs enabling tensor operations.

**Traditional Node:** PCA, K-Means, etc. excel due to the node’s memory, CPU, and on-chip bandwidth

Big Data Analytic Suite

Speedup Over Titan Baseline for CORAL-2 Big Data Benchmarks (based on pbdR)

Weak Scaling of Data Benchmarks on Titan

Strong Scaling of Data Benchmarks on SummitDev
Deep Learning Suite

Speedup Over Titan Baseline for CORAL-2 Deep Learning Benchmarks

- CANDLE
- RNN
- CNN-googlenet
- CNN-vgg
- CNN-alexnet
- CNN-overfeat

Strong Scaling of ResNet-50 on Summit

Scaling of Resnet-50 based on Keras (Tensorflow backend) and Horovod on ImageNet data
## Performance model for BDAS

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Workload (4aff10a)</th>
<th>Input size</th>
<th>SMT (thread</th>
<th>rank)</th>
<th>walltime(s)</th>
<th>8GB</th>
<th>64GB</th>
<th>Power9</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
<th>8GB</th>
<th>64GB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8GB</strong></td>
<td>PCA</td>
<td>1</td>
<td>6.8</td>
<td>3.58x</td>
<td>3.24x</td>
<td>2.83x</td>
<td>3.6</td>
<td>1.89x</td>
<td>1.71x</td>
<td>1.50x</td>
<td>3.7</td>
<td>1.95x</td>
<td>1.76x</td>
<td>1.54x</td>
<td>4</td>
<td>2.11x</td>
<td>1.90x</td>
<td>1.67x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2(2</td>
<td>40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4(2</td>
<td>40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8(4</td>
<td>40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>53.1</td>
<td>4.08x</td>
<td>4.35x</td>
<td>3.93x</td>
<td>28.2</td>
<td>2.17x</td>
<td>2.31x</td>
<td>2.09x</td>
<td>24.8</td>
<td>1.91x</td>
<td>2.03x</td>
<td>1.84x</td>
<td>25.9</td>
<td>1.99x</td>
<td>2.12x</td>
<td>1.92x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2(2</td>
<td>40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4(4</td>
<td>20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8(8</td>
<td>20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>4.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.52x</td>
<td>1.57x</td>
<td>1.57x</td>
<td>1.55x</td>
<td>1.60x</td>
<td>1.60x</td>
<td>1.55x</td>
<td>1.60x</td>
<td>1.60x</td>
<td>1.58x</td>
<td>1.63x</td>
<td>1.63x</td>
<td>2.78x</td>
<td>2.75x</td>
<td>2.74x</td>
<td>1.88x</td>
<td>1.86x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2(2</td>
<td>40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.07x</td>
<td>1.98x</td>
<td>1.98x</td>
<td>2.08x</td>
<td>1.98x</td>
<td>1.98x</td>
<td>2.08x</td>
<td>1.98x</td>
<td>1.98x</td>
<td>2.08x</td>
<td>1.98x</td>
<td>1.98x</td>
<td>2.08x</td>
<td>1.98x</td>
<td>1.98x</td>
<td>2.08x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4(4</td>
<td>20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>34.6</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8(8</td>
<td>20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>37.5</td>
<td>34.7</td>
<td>34.7</td>
<td>37.5</td>
<td>34.7</td>
<td>34.7</td>
<td>37.5</td>
<td>34.7</td>
<td>34.7</td>
<td>37.5</td>
<td>34.7</td>
<td>34.7</td>
<td>37.5</td>
<td>34.7</td>
<td>34.7</td>
<td>37.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>34.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>33.1</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
<td>33.1</td>
<td>34.7</td>
<td>33.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2(2</td>
<td>40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>286.6</td>
<td>285.7</td>
<td>285.7</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4(4</td>
<td>20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8(8</td>
<td>160)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>291.5</td>
<td>286.6</td>
<td>286.6</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
<td>283.1</td>
</tr>
</tbody>
</table>

\[
\log (Perf_{Power}) = \text{Architecture} + \text{Size} + \text{Workload} + \text{Threads}
\]
# Performance model for DL workloads

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Workload</th>
<th>Implementation</th>
<th>Precision</th>
<th>CNN WINOGRAD_NONFUSED</th>
<th>CNN IMPLICIT_PRECOMP_GEMM</th>
<th>Volta LSTM</th>
<th>Volta RNN</th>
<th>Volta GRU</th>
<th>NCCL</th>
<th>Comm</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WINOGRAD_NONFUSED</td>
<td></td>
<td>fp32</td>
<td>fp16</td>
<td>fp32</td>
<td>fp16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMPLICIT_PRECOMP_GEMM</td>
<td></td>
<td>fp32</td>
<td>fp16</td>
<td>fp32</td>
<td>fp16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.1</td>
<td>227.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NCCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>wall time (s)</td>
<td></td>
<td>2.99</td>
<td>1.98</td>
<td>5.1</td>
<td>227.98</td>
<td>0.33</td>
<td>0.46</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Workload</th>
<th>Implementation</th>
<th>Precision</th>
<th>WINOGRAD_NONFUSED</th>
<th>IMPLICIT_PRECOMP_GEMM</th>
<th>LSTM</th>
<th>RNN</th>
<th>GRU</th>
<th>NCCL</th>
<th>Comm</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WINOGRAD_NONFUSED</td>
<td>fp32</td>
<td>fp16</td>
<td>fp32</td>
<td>fp16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>IMPLICIT_PRECOMP_GEMM</td>
<td>fp32</td>
<td>fp16</td>
<td>fp32</td>
<td>fp16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSTM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.1</td>
<td>1.71x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>RNN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.1</td>
<td>1.57x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>GRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.45x</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NCCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>359.5</td>
<td></td>
<td></td>
<td>1.58x</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td>1.27x</td>
</tr>
</tbody>
</table>

Problem size:
- WINOGRAD_NONFUSED: input: 112x112x64x16 filter: 3x3x128
- IMPLICIT_PRECOMP_GEMM: input: 112x112x64x8 filter: 3x3x128
- lstm:1024-64-25 gru: 1024-64-1500 (RNN)
- 100000 4nodes (Comm)
Takeaway - ML

- Per node, expect ~2x over SummitDev, up to ~35x over Titan.
- OpenBLAS provides close performance as IBM ESSL, although ESSL seems to handle SMT better.
- Use SMT=1/2 on Summit SMT=2/4 on SummitDev for pbdR and oversubscribe threads.
- Use RAPIDS, H2O4GPU, SnapML (close source), etc to take advantage of GPUs

For more details, please refer to arXiv:1811.02287
Takeaway - DL

• Per node, expect ~2.5x over Summit-Dev, up to ~ 80x over Titan.
• Average ~60x for CNN workloads, ~20x for RNN workloads, over Titan
• ~1.5x in communication over Summit-Dev
• Near ideal scaling for Keras (Tensorflow backend) + Horovod up to 64 nodes for Resnet50 on ImageNet
IBM’s SnapML

SVM Benchmark (seconds)

- **IBM’s SnapML**
- 8GB: 17.3 (SnapML), 16.7 (Kazaam)
- 32GB: 55 (SnapML), 82.2 (Kazaam)
IBM’s SnapML

SnapML-SVM (PowerAI 1.5.3)

GB/s

Nodes

- 32GB
- 128GB
- ideal

- 40%
- 77%
Distributed deep learning

• Data parallel
  – **Synchronized**
  – Stale
  – Asynchronized

• Model parallel

• Hybrid

Review: arXiv:1802.09941
TensorFlow Resnet50 profiling on Summit

- Conv2D: 36%
- ncclAllReduce: 18%
- FusedBatchNorm: 6%
- Relu: 13%
- Others: 27%
“mini-MPI” for distributed deep learning

- NCCL (Nvidia): collective multi-GPU communication

- Horovod (Uber): Tensorflow and Pytorch support
  - NCCLReduceScatter - MPIAllreduce – NCCLAllgather for data divisible by local_rank()
  - NCCLReduce - MPIAllreduce – NCCLBcast for the remainder
  - Tensor Fusion: fuse small allreduce tensor operations into larger ones for performance gain
  - Compression (cast vars to fp16) before allreduce

- GLOO (Facebook): Pytorch support

- DDL (IBM): Tensorflow, Pytorch, Caffe support. Close source.
NCCL vs MPI allreduce

Time to transfer $10^5$ floats on SummitDev

- MPI
- NCCL
Differences in scaling up: DL VS simulation

• DL is a global optimization, changing scale -> changing solution space.
  – DL usually requires changing network architecture, update scheme, etc

• Scale in OPS ≠ Scale in time-to-solution (accuracy)
  – Tradeoff between more epochs and faster convergence

• High per-node OPS makes DL comm- and/or IO- bound at relatively small node count.
  – DL requires special designed comm (mainly all-reduce) and IO pipeline
Synchronized data parallel: scaling vs convergence

(a) Minibatch Effect on Accuracy and Performance (Illustration)

- Possible causes: “generalization gap” (Keskar et al. 2017)
  - loss of the explorative properties
  - tend to converge to sharp minimizers
  - model overfits the training data

(b) Empirical Accuracy (ResNet-50, figure adapted from [Goyal et al. 2017], lower is better)
Large mini-batch size training

• mini-batch size 8K (arXiv:1706.02677)
  – Warmup with default learning rate for optimizer
  – Start with learning rate multiplying # of workers
  – Decay learning rate periodically

• mini-batch size 32K
State-of-the-art Imagenet training

Chronology of Distributed Deep Learning Records

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Batch Size</th>
<th>Processor</th>
<th>DL Library</th>
<th>Time</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>He et al.</td>
<td>256</td>
<td>Tesla P100 x8</td>
<td>Caffe</td>
<td>29 hours</td>
<td>75.3%</td>
</tr>
<tr>
<td>2017</td>
<td>Goyal et al.</td>
<td>8K</td>
<td>Tesla P100 x256</td>
<td>Caffe2</td>
<td>1 hour</td>
<td>76.3%</td>
</tr>
<tr>
<td>2017</td>
<td>Smith et al.</td>
<td>8K → 16K</td>
<td>full TPU Pod</td>
<td>TensorFlow</td>
<td>30 mins</td>
<td>76.1%</td>
</tr>
<tr>
<td>2017</td>
<td>Akiba et al.</td>
<td>32K</td>
<td>Tesla P100 x1024</td>
<td>Chainer</td>
<td>15 mins</td>
<td>74.9%</td>
</tr>
<tr>
<td>2018</td>
<td>Jia et al.</td>
<td>64K</td>
<td>Tesla P40 x2048</td>
<td>TensorFlow</td>
<td>6.6 mins</td>
<td>75.8%</td>
</tr>
<tr>
<td>2018</td>
<td>Mikami et al.</td>
<td>34K → 68K</td>
<td>Tesla V100 x2176</td>
<td>NNL</td>
<td>224 secs</td>
<td>75.03%</td>
</tr>
</tbody>
</table>
State-of-the-art Imagenet training (arXiv:1811.05233)

- Batch size control + LARS -> 68K mini-batch size
- 2D-Torus All-reduce communication

- 224s training -> 75.03% top1 accuracy and 66% scaling efficiency on 2176 V100.
Without tuning

Resnet50 on Imagenet (Images/s)

4 nodes on Summit, batch-size=64

<table>
<thead>
<tr>
<th></th>
<th>Keras (TF backend; JPEG)</th>
<th>Pytorch (JPEG)</th>
<th>Tensorflow (TFRecord)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2205</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3642</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3926</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tuning of Tensorflow and Keras

Performance Tuning

4 nodes on Summit, batch-size=64

Preprocessing:
TF: prefetch: buffer=1,threads=4;gpu-private,threads=8
Keras: TFRecord; tf.datasets, interleave,

<table>
<thead>
<tr>
<th></th>
<th>Tensorflow</th>
<th>Keras</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3926</td>
<td>5033</td>
</tr>
<tr>
<td></td>
<td>6394</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6915</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2205</td>
<td></td>
</tr>
</tbody>
</table>

Default | I/O optimization | CPU/MEM binding
TF benchmark on Summit

TF CNN Benchmark Imagenet (TFRecord)

Difference with Synthetic data: < 5%

- batch size = 128
- ideal
- batch size = 64
Mixed precision & Tensorcore

- **Consideration**
  - Imprecise weights
  - Gradients underflow
  - Reduction overflow

- **Verification**
  - `s884cudnn`
NVProf

Synthetic Data

TFRecord
Lessons learned from Exa-scale DL on Summit (arXiv:1810.01993)

• Data ingestion (mostly coincide with TF performance guide)
  – Input pipeline, queuing input for compute
  – Concurrent processing with map

• Communication
  – Broadcast tree
    • Hierarchical aggregation of the control message (the order of tensors to be reduced)
  – Hybrid NCCL-MPI allreduce
    • NCCL intra node allreduce
    • 4 ranks (2 on each socket, b/c 4 IB devices) per node each MPI_Allreduce on a quarter of the data
    • NCCL intra node broadcast
Lessons learned from Exa-scale DL on Summit (arXiv:1810.01993)

• Algorithmic considerations
  – Weighted loss, i.e. each pixel contributes differently to the loss function, specific to application (background vs area of interest)
  – LARC, a variant on LARS, for large batch sizes.
  – Multi-channel (16), more compute, more accurate
  – Gradient lag, overlap communication and computation
  – Network, larger layer, less number of layers, to improve compute intensity.
DL vs conventional ML

• It depends.
  – In general, DL works better for unstructured features, e.g. images, text; gradient boosting works better for data with structured ones, e.g. tabulated data; feature selection + gaussian process (equivalent infinite width neural network) works better for limited data and explainability.

• Explored in several use cases.
  – Simulation energy prediction
  – Material design (High entropy alloy)
  – Climate surrogate modelling
  – Microscopic images classification
How to learn the inverse? Simulate many physically realizable solutions

Training Phase
- CBED Patterns
- Inputs → Forward Simulations → Descriptors or Model Inputs → Structural Descriptors

Testing Phase
- Experimental Data → Inverse Solution Engine (Deep Neural Network) → Structural descriptors
3D Convolutional Layers

Kernel size (2,5,5)*

Number of Filters

Dense Layers

Softmax Layer

Step/Diffuse Classification

3D Input

3D Average Pooling Layer

32

64

64

256

256

2

256 units

256 units

2 units

Along interface

Prediction: Step (p=0.96). Actual: Step

Prediction: Diffuse (p=1.00). Actual: Diffuse

100 epochs in total

validation accuracy

training accuracy
**Thickness:** 250 Å

- **Prediction:** 5 (p=0.94). **Actual:** 5
- **Prediction:** 6 (p=0.61). **Actual:** 7
- **Prediction:** 9 (p=0.56). **Actual:** 9
- **Prediction:** 2 (p=0.99). **Actual:** 2
Use Case 2 (LDRD PI: Markus Eisenbach)

- **Initial Stage**
  - **First principles calculation**
  - **State of system**
  - **MC Algos**
  - **Statistic sampling**

**Level 2**
- **Input**
- **Drives**
- **System updates**

**Level 1**
- **MC update**
- **E₁: HD-NNP**
- **E₁: NNP**

**DNN-MC Classifier**
- Proof of concept for an online model (Heisenberg)
- Offline models for complex systems (Water cluster, FeCo alloy)
- Exploration of sampling algorithms (Metropolis, Wang-Landau, Nested Sampling)

<table>
<thead>
<tr>
<th>System</th>
<th>Sampling algorithms</th>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heisenberg</td>
<td>Metropolis</td>
<td>XGBoost</td>
<td>DNN</td>
</tr>
<tr>
<td></td>
<td>Nested Sampling</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Water cluster</td>
<td>Wang-Landau</td>
<td>XGBoost</td>
<td>DNN</td>
</tr>
<tr>
<td>FeCo alloy</td>
<td>Metropolis</td>
<td>DNN</td>
<td>87%</td>
</tr>
</tbody>
</table>
Use Case 3: Alloy design
(with Zongrui Pei)

Joint experiment and simulation

Mechanism I: source of dislocations

Descriptor I: YSI ($\varepsilon_{SFE}$) evaluation

Machine Learning evaluation

Descriptor II: $\Delta E^{I-II}$ ($\delta_x$)

Mechanism II: mobility of dislocations

Simulations

Ductile Solutes

The 22 elemental properties

Stacking fault energies

Stacking fault energies
Questions?