Scalable Distributed Training of Large Neural Networks with LBANN

Naoya Maruyama
maruyama3@llnl.gov

7th Annual MVAPICH User Group Meeting
August 20, 2019
Acknowledgement

- Nikoli Dryden (UIUC / LLNL)
- Yosuke Oyama (Tokyo Tech / LLNL)
- Brian Van Essen (LLNL)
- Tom Benson (LLNL)
- Tim Moon (LLNL)
- LBANN Team
Training Deep Convolutional Neural Networks

- Iterative computations until error rate becomes sufficiently small
- Each iteration is a statically fixed DAG of matrix computations
  - Convolutions
  - Matrix multiplications
  - Element-wise filtering
  - Reductions
- Training sweeps a large collection of labeled samples by picking up a subset of samples ("mini-batch")

```plaintext
while unconverged
    pick a mini-batch
    traverse DAG from input to output
    traverse DAG from output to input
    adjusts network parameters
endwhile
```

Aphex34 [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], from Wikimedia Commons
LBANN: Livermore Big Artificial Neural Network Toolkit

- [https://github.com/LLNL/lbann](https://github.com/LLNL/lbann)

- Deep Neural Network training / classification
  - Optimized distributed memory algorithms
    - Including spatially decomposed convolutions
  - Optimized asynchronous communication library
  - Compose parallelism at multiple levels
  - Optimize for strong & weak scaling

- Unique HPC resources at scale
  - InfiniBand or Omnipath interconnect
  - Tightly-coupled GPU accelerators
  - Node-local NVRAM
  - High bandwidth Parallel File System
  - State-of-the art distributed linear algebra library
Parallel Training is Critical to Meet Growing Compute Demand

- Training is an extremely compute-intensive task
- And growing exponentially
  - Doubling every 3.5 months
- Distributed training is essential and is proven to be successful

### Table 1: Large-scale ResNet-50 training results.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Chips</th>
<th>Batch</th>
<th>Optimizer</th>
<th>BN</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goyal et al. [6]</td>
<td>P100</td>
<td>256</td>
<td>8192</td>
<td>Momentum Local</td>
<td>76.3%</td>
<td>1 hour</td>
</tr>
<tr>
<td>Smith et al. [16]</td>
<td>TPU v2</td>
<td>128</td>
<td>8192 → 16384 Momentum Local</td>
<td>76.1%</td>
<td>30 mins.</td>
<td></td>
</tr>
<tr>
<td>Akiba et al. [2]</td>
<td>P100</td>
<td>1024</td>
<td>32768</td>
<td>RMS + Mom. Local</td>
<td>74.9%</td>
<td>15 mins.</td>
</tr>
<tr>
<td>Jia et al. [10]</td>
<td>P40</td>
<td>1024</td>
<td>65536</td>
<td>LARS Local</td>
<td>76.2%</td>
<td>8.7 mins.</td>
</tr>
<tr>
<td>Baseline</td>
<td>TPU v2</td>
<td>4</td>
<td>1024</td>
<td>Momentum Local</td>
<td>76.3%</td>
<td>8.0 hours</td>
</tr>
<tr>
<td>Ours</td>
<td>TPU v2</td>
<td>256</td>
<td>16384</td>
<td>Momentum Local</td>
<td>75.1%</td>
<td>10 mins.</td>
</tr>
<tr>
<td>Ours</td>
<td>TPU v2</td>
<td>256</td>
<td>32768</td>
<td>LARS Local</td>
<td>76.3%</td>
<td>8.5 mins.</td>
</tr>
<tr>
<td>Ours</td>
<td>TPU v3</td>
<td>512</td>
<td>32768</td>
<td>LARS Local</td>
<td>76.4%</td>
<td>3.3 mins.</td>
</tr>
<tr>
<td>Ours</td>
<td>TPU v3</td>
<td>1024</td>
<td>32768</td>
<td>LARS Distributed</td>
<td>76.3%</td>
<td>2.2 mins.</td>
</tr>
</tbody>
</table>

Ying et al., “Image Classification at Supercomputer Scale,” Systems for ML Workshop @ NIPS 2018

Generalized Parallel Convolution in LBANN

Sample (Data) Parallelism
- Allreduce
- O(100-1000) GPUs
- NCCL/MPI

Spatial Parallelism
- Halo exchange
- O(10) GPUs
- MPI/Custom

Channel/Filter Parallelism
- Allgather/ReduceScatter
- O(10) GPUs
- NCCL/MPI
The DOE has large scientific data sets that are unlike any commercial data set:
- May not be natural images
- Large samples
- May be generated by computational simulations

Even a mini-batch with just one sample may require more than $O(10)$ GB of memory \(\rightarrow\) Does not fit in a single GPU memory.

### Mesh Tangling Detection
- $O(1000)^2$ mesh

### Cosmological Analysis
- $O(100)^3$ volumetric data

[Mathuriya18]
10x Better Prediction Accuracy with Large Samples

- **Dataset:** cosmoUniverse_2019_05_4pare
  - [https://portal.nersc.gov/project/m3363/cosmoUniverse_2019_05_4parE](https://portal.nersc.gov/project/m3363/cosmoUniverse_2019_05_4parE)
- **CosmoFlow model** [Mathuriya18] + batchnorm
- **Prediction of** Omega_m
Scaling Performance beyond Data Parallel Training

1024^2 training samples

Lassen: Power 9 x 2 + Volta 100 x 4

Strong Scaling Performance

Mini-batch size: 128; 128 GPUs to 2048 GPUs

>4x faster than the standard parallel method
Parallel Training

- Parallelism lies inside the training loop

- Parallel convolution
  - $N$: number of samples, $C$: number of channels, $F$: number of filters, $H$: height, $W$: width
  - Input: images of $N \times C \times H \times W$
  - Filters: $F \times C \times K \times K$
  - Output: $N \times F \times H \times W$
  - Partitioning the samples is the most common approach

Not parallelizable with standard SGD

```
while un converged
  pick a mini-batch
  traverse DAG from input to output
  traverse DAG from output to input
  adjusts network parameters
endwhile
```

Images
$N \times C \times H \times W$

Filters
$F \times C \times K \times K$

Feature maps
$N \times F \times H \times W$
Sample-Parallel Convolution

- Split the samples of a mini-batch between processes (or GPUs)
- Each process computes independently with reductions of gradients
- Implemented in many of DL frameworks such as LBANN, TensorFlow, PyTorch, Chainer, etc.

![Diagram of Sample Parallelism and Spatial Parallelism]

Images: \( N \times C \times H \times W \)
Filters: \( F \times C \times K \times K \)
Feature maps: \( N \times F \times H \times W \)
Scalability Limitations of Sample-Parallel Training

- **Limited memory capacity**
  - Neural networks are becoming deeper
  - Sample size is becoming bigger when dealing with scientific data rather than cats and dogs (^.^.)
  - Does not fit GPU memory

- **Limited parallelism**
  - The degree of parallelism depends on the number of samples in a mini-batch (i.e., $N$)
  - Can’t make $N$ arbitrary large as learning accuracy significantly drops
  - Usually, $N$ is $O(100)$-$O(1000)$

---

Goyal et al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2017.
Parallelism is not Limited to the Sample Dimension

Partitioning a mini-batch (*Sample Parallelism*)

Images
\[ N \times C \times H \times W \]

Filters
\[ F \times C \times K \times K \]

Feature maps
\[ N \times F \times H \times W \]

Partitioning each sample (*Spatial Parallelism*)
Boundary Data Exchange

- Convolution/pooling needs adjacent data for each point \( \rightarrow \text{Halo exchange} \)
- Halo size depends on filter size and number of channels
- Also on stride and dilation
  - Does not depend on grouped convolution
- Pooling involves a “reverse halo exchange”
  - Push/accumulate values to remote
### Hybrid Parallel Convolution

- Sample and spatial parallelisms are orthogonal and can be used simultaneously.
- Can use $P_N \times P_S$ GPUs.
- If $N > \#GPUs$ && sizeof(activations+weights) < GPU memory size
  - Use sample parallel only.
- Otherwise:
  - Spatial parallel to fit GPU memory
  - Sample parallel up to $N$.
- Example:
  - Spatial parallel with intra-node 4 GPUs of Sierra nodes
  - Sample parallel with as many nodes as the mini-batch size.

**Images**

$\times \# \times \$ \times \%

**Feature maps**

$N \times F \times H \times W$

**Filters**

$F \times C \times K \times K$

**Images**

$N \times C \times H \times W$

An example case with nested partitioning along sample and spatial domains.
Extend the LBANN toolkit (https://github.com/LLNL/lbann) — Annotate layers with parallel execution strategies

Distconv: Custom distributed tensor library for convolution — Similar approach as stencil libraries

Aluminum for communication (https://github.com/LLNL/Aluminum) — CUDA peer-to-peer intra-node — Aluminum’s custom MPI+CUDA backend for inter-node — Collective routines with MPI and NCCL

NVIDIA cuDNN for CNN GPU kernels
Optimizing Halo Exchange

- Standard MPI + CUDA communication model is inefficient for CNNs
  - GPU stream synchronization before MPI_Send is mandatory
  - GPU stream synchronization can be very costly
  - Can’t keep issuing asynchronous GPU tasks, causing GPU kernel launch overhead (~10 us)

- Not an issue when doing weak scaling
  - Common in scientific computing

- Only strong scaling is possible with CNNs

Sending data from GPU:
```c
do_something<<<,,,st>>>(...)
(cudMemcpyDeviceToHost)
cudaStreamSynchronize(st)
MPI_Send()
do_something<<<,,,st>>>(...)
```
Asynchronous Inter-Node Data Transfer

- CUDA stream memory operations allow GPU data transfer without GPU synchronization
- No GPU synchronization
- Implemented in the *Aluminum* communication library

```c
Main thread

do_something<<<,,st>>>(...)
(cudMemcpyDeviceToHost)
cudaEventRecord(ev, st)
cuStreamWaitValue32(st, ...)
do_something<<<,,st>>>(...)

Comm thread

while (cudaEventQuery(ev)) {}  
MPI_Sendrecv()
(cudMemcpyHostToDevice)
cuStreamWriteValue32(st,...)
```
Aluminum: GPU-Centric Communication Library

- Generic C++ interface to MPI, NCCL, custom algorithms
  - Asynchrony via dedicated communication engine

- Communication is “just another kernel” (GPU-centric, like NCCL)
  - Runtime handles all details (like non-blocking MPI—“progress is magic”)

- Associate a “stream of computation” with a communicator
  - CUDA stream; implicit on CPU, but could be a (lightweight) thread, ...
  - Aluminum ensures communication does not begin until data on the stream is ready

- Blocking operations (like MPI_Allreduce):
  - Ensure no subsequent operations start until communication complete
  - Block only the associated stream

- Non-blocking operations (like MPI_Iallreduce):
  - Block no stream
  - Explicit completion operation (like MPI_Wait)

- https://github.com/llnl/aluminum
// Synchronous MPI:
for (int step = 0; step < num_steps; ++step) {
    load_mini_batch();
    for (auto&& layer : layers) layer.forward();
    for (auto&& layer : layers) {
        layer.backprop_data();
        if (layer.has_weights()) {
            layer.backprop_filter();
            cudaStreamSynchronize(stream);
            MPI_Allreduce(MPI_IN_PLACE, layer.weights, layer.size, MPI_FLOAT, MPI_SUM, comm);
            layer.sgd_step();
        }
    }
}

// Aluminum:
for (int step = 0; step < num_steps; ++step) {
    load_mini_batch();
    for (auto&& layer : layers) layer.forward();
    for (auto&& layer : layers) {
        if (layer.has_weights()) {
            layer.backprop_filter();
            Al::NonblockingAllreduce<Al::MPICUDABackend>(
                layer.weights, layer.size,
                Al::ReductionOperator::sum, comm, layer.req);
        }
        layer.backprop_data();
    }
    for (auto&& layer : layers) {
        Al::Wait<Al::MPICUDABackend>(layer.req);
        layer.sgd_step();
    }
}
Evaluation

- **Lassen - 700 nodes**
  - 2x POWER9 + 4X V100 + NVLink2 + 2x InfiniBand EDR
  - Experiments use up to 2048 GPUs

- **cuDNN v7.5.1**

- **MVAPICH2-GDR 2.3.2**

- **Spectrum MPI/2019.01.30**

- **Mesh tangling data:**
  - 1K: 1024 x 1024 x 18
  - 2K: 2048 x 2048 x 18

- **Same spatial distribution for all layers**
Performance of Spatial-Parallel Convolution

Convolution of a single $1024^2$ image with 16 channels and 16 filters of 3x3 kernels

1GPU: 0.533 ms

- MV2-GDR is faster than Spectrum MPI
- Aluminum is faster than MV2-GDR
- Up to 5x speedup
End-to-End Hybrid Parallel Training of Mesh Model

- **2048^2 mesh tangling model**
  - Needs 2 GPUs per sample at least

- **Weak scaling with sample parallelism**
  - Excellent scaling by overlapped gradient allreduces
  - Small increase at 2048 GPUs due to decreased work to hide allreduces

- **Strong scaling with spatial parallelism**
  - Close to 4x with 16 GPUs
  - Speedup is smaller at later layers due to decreasing spatial dimensions

![Graph showing 2048x2048 mesh model scaling](image)
Conclusion

- Generalized parallel convolution
  - Sample/spatial/channel/filter parallelism
  - Address memory capacity constraint → Enables more accurate models
  - Allow strong scaling → Faster training time

- LBANN: Scalable deep learning software stack for large-scale science and engineering problems
  - https://github.com/llnl/lbann
  - Aluminum: GPU-centric communication library (https://github.com/llnl/aluminum)
  - Implements the generalized parallel convolution algorithms
Ongoing Work

- Channel/filter parallelism
  - Early results to appear at SC19

- Performance modeling to identify optimal parallelization strategies

- Optimizing performance of reading training samples

References