Single- and Multiple-Machine Training with MXNet

Carl Yang

Mentors: Rahul Huilgol, Haibin Lin
Amazon AWS AI, Palo Alto

October 4, 2018
Deep learning

- Forward propagation
- Backward propagation
- Weight update
Deep learning with multiple GPUs

- Forward propagation
- Backward propagation
- All-Reduce (add dense matrices together elementwise)
- Weight update
Challenges

- Network between GPUs and machines are complex
- The topology of the network and bandwidth of links can vary widely
A Tale of Two Paradigms

Single-machine Training:

- NVLink network (300GB/s)
- fixed topology
- low # of processes (8)

Multi-machine training:

- Ethernet network (25Gb/s)
- arbitrary topology
- high # of processes (32+)

Q: How do we take advantage of these differences?
Format of Presentation

1. Introduction
2. Single-machine training
3. Multi-machine training
4. Conclusion
Format of Presentation

1. Introduction
2. Single-machine training
3. Multi-machine training
4. Conclusion
Single-Machine Training

- NVLink network (300GB/s)
- fixed topology
- low # of processes (8)

- take advantage of topology info [1]
- saturate links by slicing large gradients

Latency vs. Message Size

![Latency vs. Message Size Graph](image.png)

- **Latency-bound**
- **Ideal**
- **Bandwidth-bound**

Runtime (s)

Message Size (words)
Slice Large Gradients for Better Performance
Related Work

Nvidia NCCL: ring algorithm

--kv-store device: parameter server on single machine (PS-single)
Our Solutions

Single Tree

Multi Tree
Design: Single-Machine Topology-Aware Communication

1. Detect topology at runtime
2. Build trees using topology information and constraints:
   a. binary
   b. maximum weight
   c. minimum height
3. Execute KVStore::Push and KVStore::Pull using tree

Now available in MXNet 1.3

Try it out by setting environmental variable MXNET_KVSTORE_USETREE=1
3. Execute KVStore::Push and KVStore::Pull using tree
p3.16xlarge
Step 1: Send gradient along red arrow and reduce
Step 2: Send partial sum along red arrow and reduce
Step 3: Send partial sum to GPU 5 and reduce
KVStore::Push complete: Reduced gradient is at GPU 5
Experiments and Results

![VGG-16 fp32 Graph](image-url)

- **X-axis**: Batch Size (per GPU)
- **Y-axis**: Speed-up vs. NCCL
- **Legend**:
  - Parameter Server
  - Exhaustive Search
  - Kernighan-Lin

The graph above illustrates the speed-up comparison between different batch sizes for VGG-16 fp32 in terms of NCCL. Each bar represents the speed-up for a specific batch size compared to the baseline. The comparison is made among Parameter Server, Exhaustive Search, and Kernighan-Lin methods.
Experiments and Results

ResNet-50 fp32

Parameter Server  Exhaustive Search  Kernighan-Lin

Speed-up vs. NCCL

Batch Size (per GPU)

4  8  16  32  64
Takeaway: Different models imply different gradient distributions imply different computation/communication

VGG (2014)
16 GFlops
138M words

4 GFlops
25M words
### Topology and Slice Large Gradients Both Contribute

Model: VGG-16  
Batch size: 4 per GPU  
Hardware: 8 V100s on p3.16xlarge

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Performance (samples/sec)</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (kv-store device)</td>
<td>123</td>
<td>-</td>
</tr>
<tr>
<td>Topology-awareness</td>
<td>358</td>
<td>2.91x</td>
</tr>
<tr>
<td>Slice large gradients</td>
<td>724</td>
<td>2.02x</td>
</tr>
</tbody>
</table>
Format of Presentation

1. Introduction
2. Single-machine training
3. Multi-machine training
4. Conclusion
Two types of distributed training

Parameter server

Allreduce
Horovod

- Open-source third-party distributed training framework for TensorFlow, Keras and PyTorch made by Uber
- Paper [1] claims better scalability than TensorFlow’s built-in distributed training

Design: Horovod-MXNet Integration

Diagram:

- Horovod
  - DistributedOptimizer
    - `hvd.allreduce`
    - `horovod_mxnet_allreduce_async`
      - `DoAllreduceAsync`
        - Horovod Backend Stuff
  
- MXNet
  - Optimizer
    - mxnet::Engine::PushAsync()
Single-Machine Training

- NVLink network (300GB/s)
- fixed topology
- low # of processes (8)

Multi-Machine Training

- Ethernet network (25Gb/s)
- arbitrary topology
- high # of processes (32+)

Tensor Fusion Improves Scalability from 25% to 68%
Q: Why Does Tensor Fusion Get Better Performance

The graph shows the runtime (s) against the message size (words) for latency-bound, ideal, and bandwidth-bound scenarios. Tensor fusion is marked as a point of interest, indicating where it improves performance compared to the other scenarios.
Can: What if we combine NCCL+MPI?

- **Intra-node**: NCCL
  - topology-aware algorithm
  - low # of processes (8)
- **Inter-node**: MPI
  - latency-optimal algorithm $O(lg p)$
  - high # of processes (32+)

Kudos to Can Karakus, TensorFlow team for implementing this capability!
Hierarchical Allreduce Boosts Scalability 68% to 91.2%
Comparison with Parameter Server (91.2% vs. 68%)
Future Work

- I will submit PRs to Horovod and MXNet repos
- Potential issue: Horovod+MXNet non-hierarchical (NCCL only) scalability is lagging Horovod+TensorFlow
  - Poor interaction between NCCL and MXNet engine?
- Horovod+SageMaker
- Horovod+Elastic Training
- Horovod+Keras+MXNet
- Horovod+Sparse support
- Horovod+Gluon support
Q: How do we take advantage of these differences?

**Single-Machine Properties**
- NVLink network (300GB/s)
- fixed topology
- low # of processes (8)

**Multi-Machine Properties**
- Ethernet network (25Gb/s)
- arbitrary topology
- high # of processes (32+)

**Strategy**
- take advantage of topology info
- saturate links by slicing large gradients

**Implementation**
- NCCL
- KVStore tree

**Strategy**
- batch tensors since only one message can be sent
- use latency-optimal MPI algorithms instead of NCCL ring algorithm

**Implementation**
- MPI
Conclusion

- Differences between single- and multi-machine interconnects imply different approaches must be taken:
- Single-machine: Topology-awareness and gradient slicing are two important techniques that get peak 6.6x speed-up compared to built-in MXNet communication method
- Multi-machine: Tensor fusion is important
  - scalability: 25% -> 68% at 256 GPUs
- Differences imply hierarchical allreduce is a good idea, because it combines the best of both worlds
  - scalability: 68% -> 92.1% at 256 GPUs
Acknowledgments

- My mentors Rahul and Haibin for making my project possible
- Everyone who came to my design reviews: Mu, Eric (Junyuan), Leyuan, Andrea, Steffen, Hagay, Can, Max, Hao, Jun, Deep Engine Team
Questions?

Contact info:

Carl Yang, ctcyang@ucdavis.edu
Experiments and Results

AlexNet fp32

<table>
<thead>
<tr>
<th>Batch Size (per GPU)</th>
<th>Speed-up vs. NCCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1.2</td>
</tr>
<tr>
<td>8</td>
<td>1.4</td>
</tr>
<tr>
<td>16</td>
<td>1.6</td>
</tr>
<tr>
<td>32</td>
<td>1.4</td>
</tr>
<tr>
<td>64</td>
<td>1.2</td>
</tr>
<tr>
<td>100</td>
<td>1.0</td>
</tr>
</tbody>
</table>

- Parameter Server
- Exhaustive Search
- Kernighan-Lin
Q: How do we overlap computation/communication?

- Comp: backpropagation of layer n
- Comm: allreduce of layer n
- Comp: backpropagation of layer n-1
- Comm: allreduce of layer n-1
- ...

![Diagram of forward and backward propagation in a neural network.](image-url)
Idea #1: Deep learning framework automatically takes advantage of computation/communication overlap

All popular deep learning frameworks (TensorFlow, PyTorch and MXNet) have a dependency engine responsible for:

- Building task graph of required operations
- Scheduling these operations
- Executing independent operations concurrently
Idea #1: Deep learning framework automatically takes advantage of computation/communication overlap

Engine treats communication as just another operator

- Type and number of request threads vary by service
- 8 threads typical (except Python, Node.js)
- Asynchronous or blocking
- C++
  - Single thread only, not configurable
  - Some calls blocking, some calls asynchronous
- C++
  - 2 by default, but configurable
  - Asynchronous calls
- CUDA
  - BLAS Libraries
  - Blocking calls