Scalable Distributed Training of Recommendation Models: An ASTRA-SIM + NS3 case-study with TCP/IP transport

Saeed Rashidi*, Pallavi Shurpali†, Srinivas Sridharan†, Naader Hassani†, Dheevatsa Mudigere†, Krishnakumar Nair†, Misha Smelyanski† and Tushar Krishna*

*Georgia Institute of Technology
†Facebook
Distributed Training

• Training a neural network involves using a training dataset to update the model weights to create a good mapping of inputs to outputs.

• Training time is increasing:
  • DNN Networks are becoming bigger (e.g. GNMT, BERT)
  • Training samples are becoming larger (e.g. DLRM)
  • Moore’s law is dead!

• Solution?
  • **Distributed training**: scale the training across more compute nodes
Challenges with Distributed Training

• Communication!
  • Inevitable in any distributed algorithm

• What does communication depend on?
  • synchronization scheme: synchronous vs. asynchronous.
  • parallelism approach: data-parallel, model-parallel, hybrid-parallel.

• Is it a problem?
  • Depends ... can we hide it behind compute?
Background: Sync. vs. Async. Training

- Defines when nodes should exchange data
  - Affects convergence time

We focus on synchronous training
Background: Data-Parallel Training

- Distribute Data across multiple nodes and replicate model (network) along all nodes.
- **No communication** during the forward pass.

Flow per layer:
1. Compute output
2. Go to the next layer

Forward pass

Inference
Communicate
Background: Data-Parallel Training

• Distribute Data across multiple nodes and replicate model (network) along all nodes.

• **Communicate weight gradients** during the backpropagation pass.
  • Blocking wait during forward pass for collective of previous backpropagation for that layer.

Flow-per-layer: 1. Compute weight gradient - 2. issue weight gradient comm - 3. compute input gradient - 4. go to previous layer
Background: Model-Parallel Training

- Distribute Model across all nodes and replicate data along all nodes.
- **Communicate outputs** during the forward pass.

Background: Model-Parallel Training

• Distribute Model across all nodes and replicate data along all nodes
• **Communicate input gradients** during the backpropagation pass.

Background: Hybrid parallel

- Partition nodes into groups. Parallelism within a group is model-parallel, across the groups is data-parallel, or vice versa.

<table>
<thead>
<tr>
<th>Parallelism</th>
<th>Activations during the forward pass</th>
<th>Weight gradients</th>
<th>Input gradients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Hybrid</td>
<td>partially</td>
<td>partially</td>
<td>partially</td>
</tr>
</tbody>
</table>
Communication during Distributed Training

• Distributed Training introduces "Collective Communication"
  • All-Reduce
    • Reduce-Scatter + All-Gather
  • All-to-All
  • One of these or a combination of these can occur depending on the DNN Model and Parallelization Strategy (Model/Data/Hybrid)

• Research Questions
  • What determines the runtime for a collective?
  • What is the compute-communication ratio during Distributed Training?
Example: Ring Based All-Reduce

• A ring with N nodes partitions data to N messages
• Collective Communication Flow:

\[
\begin{align*}
X_s^{(0)} & \rightarrow X_s^{(1)} \rightarrow X_s^{(2)} \rightarrow X_s^{(3)} \rightarrow \sum X_i^0 \\
\sum X_i^0 & \rightarrow X_i^0 \rightarrow X_i^1 \rightarrow X_i^2 \rightarrow X_i^3 \\
\sum X_i^0 & \rightarrow X_i^0 \rightarrow X_i^1 \rightarrow X_i^2 \rightarrow X_i^3 \\
\sum X_i^0 & \rightarrow X_i^0 \rightarrow X_i^1 \rightarrow X_i^2 \rightarrow X_i^3
\end{align*}
\]
Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:
Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:

Reduce-Scatter done!
Example: Ring Based All-Reduce

• A ring with N nodes partitions data to N messages
• Collective Communication Flow:
Example: Ring Based All-Reduce

- A ring with $N$ nodes partitions data to $N$ messages
- Collective Communication Flow:
Example: Ring Based All-Reduce

- A ring with N nodes partitions data to N messages
- Collective Communication Flow:

All-Gather done!
Hierarchical all-reduce:
- Reduce-scatter within package
- All-reduce across rows
- All-reduce across columns
- All-gather within package

Similar to Google TPU

Hierarchical all-reduce:
- Reduce-scatter within package
- All-reduce across switch
- All-gather within package

Similar to NVIDIA DGX2
How to Model and Evaluate the Communication Effect

- It is a complex problem and can be viewed as three layers:
  - 1. Workload layer (the training loop):
    - Parallelism approach
    - Compute power
    - Communication size & type and dependency order
  - 2. System layer:
    - Collective communication algorithm
    - Chunk size, schedule of collectives
  - 3. Network layer:
    - Physical topology
    - Congestion control, communication protocol
    - Link BW, latency, buffers, routing algorithm

Many tools in this area (e.g., Garnet, NS3)

Not too many tools cover these aspects
ASTRA-SIM Architecture: Current

• Workload layer:
  • Supports Data-Parallel, Model-Parallel, Hybrid-Parallel training loops
  • Easy to add new arbitrary training loops

• System:
  • Ring based, Tree-based, AlltoAll based, and multi-phase collectives
  • Easy to add new collective communication

• Network:
  • Supports NS3 and GARNET Network simulator
  • NS3:
    • Supports switch-based topologies
    • Supports TCP and ROCE communication protocol
  • GARNET:
    • Supports switch-based and torus-based topologies
    • Supports credit-based flow control
  • Can add new topologies in both NS3 and GARNET

S. Rashidi (GTech), S. Sridharan (Facebook), S. Srinivasan (Intel), and T. Krishna (GTech), “ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms”, ISPASS 2020
Why Simulation?

- Model systems (hardware) that do not exist today
- Analytical modeling insufficient due to actual network congestion
  - Consider a flat 128-node system with a single all-reduce collective
  - Difference could be up to 50X!

Single all-reduce example:
Simulation Methodology

- Compute model: V100 GPUs
- Network Model: NS3 backend with TCP/IP protocol.
- DNN Model: DLRM
- Target Systems: Flat100G, Flat800G, Hier, HierOpt

<table>
<thead>
<tr>
<th>Variable Parameters</th>
<th>Default Value</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Batch Size</td>
<td>65536</td>
<td></td>
</tr>
<tr>
<td>GPU Model</td>
<td>V100</td>
<td></td>
</tr>
<tr>
<td>Number of GPUs</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Compute Power Per GPU</td>
<td>60 TFLOPS</td>
<td></td>
</tr>
<tr>
<td>Per GPU batch size</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>Scale-out link BW</td>
<td>100 Gb/s</td>
<td>{100, 800}</td>
</tr>
<tr>
<td>Scale-out switch buffer</td>
<td>1GB</td>
<td>{64MB, 1024MB}</td>
</tr>
<tr>
<td>Link latency + TCP Ack delay</td>
<td>0.5us</td>
<td></td>
</tr>
<tr>
<td>Injection latency</td>
<td>0.01usec</td>
<td>{0.01, 1, 10, 100}</td>
</tr>
<tr>
<td>TCP window size</td>
<td>0.5MB</td>
<td>{0.5, 0.1}</td>
</tr>
<tr>
<td>all-reduce algorithm</td>
<td>DBT</td>
<td>{DBT, A2A}</td>
</tr>
<tr>
<td>Number of chunks</td>
<td>1024 (flat), 128 (hierarchical)</td>
<td>{1024, 128, 32}</td>
</tr>
<tr>
<td>Chunk parallelism</td>
<td>64</td>
<td>{64, 128}</td>
</tr>
<tr>
<td>Collective Scheduling</td>
<td>LIFO</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Parameters</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of embedding data-type</td>
<td>16</td>
<td>bits</td>
</tr>
<tr>
<td>Pooling factor</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Top MLP layers</td>
<td>10+2</td>
<td></td>
</tr>
<tr>
<td>Bottom MLP layers</td>
<td>5+2</td>
<td></td>
</tr>
<tr>
<td>Dense features</td>
<td>1600</td>
<td></td>
</tr>
<tr>
<td>Top MLP layer size</td>
<td>2048</td>
<td></td>
</tr>
<tr>
<td>Bottom MLP layer size</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>Sparse features</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Embedding dimension</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

Experiments Setup

DLRM Specification
Target Systems

Flat100G
1-phase all-reduce

Hier
2-phase all-reduce

HierOpt
3-phase all-reduce
Results at First Glance (Raw Latency)

Observations:

- LIFO scheduling effect
- MLP size effect
- Flat/Hierarch A2A
- Fwd/bckprop A2A
- Flat/Hierarch All-Reduce
Results at First Glance (End-to-End Latency)

Observations:

• MLP Top0

• A2A exposed latency compared to bottom MLP
Results at First Glance (Ratio Latency)

Observations:

• 3X difference in total iteration time
• Flat vs. Hierarch tradeoffs
Effect of Memory Copies

Observations:

• More sensitivity of Flat to high mem copy latencies.
Effect of Global Switch Buffer Size

Observations:

• Flat vs. Hierarch different Sensitivity to global switch size
Effect of Concurrency & Pipelining

Observations:

- Optimal point requires a balance between link utilization and congestion/queue delay times.
ASTRA-SIM Architecture: Next Version......

- Adding the online compute API interface for more accurate modeling of shared resource congestions (e.g. memory congestions) between the compute and network tasks

https://github.com/astra-sim/astra-sim
Conclusion

• The design space to build the best HW/SW platform is quite large.
• It is hard (or sometimes impossible) to change these parameters in real systems.
• Analytical model can be misleading and is not accurate.
• Our ASTRA-SIM + NS3 simulation methodology provides a convenient way to explore this space.
• We analyzed the FB DLRM Model in different systems and for Flat vs. Hierarchical topologies. Their relative performance is dependent on other factors such as: link BW, concurrency, switch buffer, algorithm selection, etc...
Thank you!