TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches

Aashaka Shah\textsuperscript{1}, Vijay Chidambaram\textsuperscript{1,2}, Meghan Cowan\textsuperscript{3}, Saeed Maleki\textsuperscript{3}, Madan Musuvathi\textsuperscript{3}, Todd Mytkowicz\textsuperscript{3}, Jacob Nelson\textsuperscript{3}, Olli Saarikivi\textsuperscript{3}, Rachee Singh\textsuperscript{4,5}

\textsuperscript{1}UT Austin \quad \textsuperscript{2}VMWare Research \quad \textsuperscript{3}Microsoft Research \quad \textsuperscript{4}Microsoft \quad \textsuperscript{5}Cornell University

Email: aashaka@utexas.edu
Distributed Deep Learning

Deep learning models are getting larger
• Distributed across various nodes/servers, each with multiple GPUs

Incurs network communication overhead
• GPUs can spend as much as 20% - 65% of time idle waiting on network communication

1. https://huggingface.co/blog/large-language-models
Communication in Distributed ML

- MPI-style collective communication used as abstractions for communication
  - Gather, shuffle, accumulate data
  - AllGather, AlltoAll, ReduceScatter, AllReduce

- Collective algorithm determines network utilization and speed of communication
Challenges in building algorithms for collectives?
Wide variety of node topologies

⇒ Best collective algorithm could be different for different topologies

Number of GPUs (8, 16)
Number of NICs (shared/dedicated)
GPU interconnects (NVLink, NVSwitches)
Challenges in building algorithms for collectives?

Hardware heterogeneity

- Link connections are heterogeneous
  - Inter-node bandwidth < intra-node bandwidth
  - Inter-node latency > intra-node latency

⇒ Efficient collective algorithms need to be built keeping in mind link heterogeneity
Challenges in building algorithms for collectives?

Data size awareness

- Collective algorithms with many data transfer steps (like a state-of-the-art Ring algorithm) perform poorly for small data sizes

⇒ Efficient collective algorithms depend on size of data chunks to be transferred
Current state-of-the-art
NCCL (NVIDIA Collective Communication Library)

[Topology awareness] Generic algorithms like Ring or Tree mapped onto target topology
- Not custom-built for a particular heterogeneous topology

[Data size awareness] Tuning to select algorithms (Ring, Tree) based on input size
- Complicated
- Not present for all collectives
- Done using experiments that may not match reality

[Availability at scale]
+ Scales to multi-node topologies

Current state-of-the-art

Synthesis-based approaches (Blink, SCCL)

[Topology and data size awareness] Synthesize collective algorithm targeted to a particular topology
  + Maximize link utilization in heterogeneous topology (Blink)
  + Synthesize pareto-optimality in terms of latency and bandwidth (SCCL)

[Availability at scale] Synthesis is NP-hard
  - Cannot scale synthesis to a multi-node topology

TACCL

Collective Communication Library that solves a set of mixed integer linear programming problems to synthesize collective algorithms

- Topology-aware
- Input-size aware

Scales to multi-node topologies

Drop-in replacement for NCCL

TACCL algorithms outperform NCCL by up-to 6.7x for evaluated topologies and provide up-to 2.4x end-to-end speedup for evaluated ML models
TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches

Target Collective → Profiled Topology → Communication Sketches → Hyperparameters → Synthesizer → Algorithm → Backend

Inputs
TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches

- Target Collective
- Profiled Topology
- Communication Sketches
- Inputs

Steps:
1. Profiled Topology
2. Communication Sketches
3. Hyperparameters
4. Synthesizer
5. Algorithm
6. Backend
What does a synthesized collective algorithm look like?

**Precondition**

<table>
<thead>
<tr>
<th>Link</th>
<th>Chunk sent?</th>
<th>Send order</th>
<th>Send time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>0</td>
<td>0</td>
<td>[0]</td>
</tr>
<tr>
<td>(1,0)</td>
<td>1 2 3</td>
<td>1 &gt; 2 = 3</td>
<td>[0, 1, 1]</td>
</tr>
<tr>
<td>(1,2)</td>
<td>0 1 3</td>
<td>1 &gt; 0 &gt; 3</td>
<td>[1, 0, 2]</td>
</tr>
<tr>
<td>(2,1)</td>
<td>2</td>
<td>2</td>
<td>[0]</td>
</tr>
<tr>
<td>(1,3)</td>
<td>0 1 2</td>
<td>1 &gt; 0 &gt; 2</td>
<td>[1,0,2]</td>
</tr>
<tr>
<td>(3,1)</td>
<td>3</td>
<td>3</td>
<td>[0]</td>
</tr>
</tbody>
</table>

**Postcondition**

L links, C chunks \(\text{(NP-hard)}\) 😞

For each link:
1. Will a data chunk be sent across it? \(O(2^{(C \times L)})\)
2. How will chunks be ordered wrt each other? \(O(2^{(C \times C \times L)})\)
Communication Sketches

- ML engineer provides *Communication Sketches*
- Specify intuitive parts of the algorithm
- Do not require a lot of domain knowledge to write
- Guide algorithm synthesis

<table>
<thead>
<tr>
<th>Links</th>
<th>Chunks</th>
<th>Sketch</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,1)</td>
<td>Y/N?</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(1,0)</td>
<td>Y/N?</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(1,2)</td>
<td>Y/N?</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(2,1)</td>
<td>Y/N?</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(1,3)</td>
<td>Y/N?</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(3,1)</td>
<td>Y/N?</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(0,1)</td>
<td>-</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(1,0)</td>
<td>-</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(1,2)</td>
<td>Y/N?</td>
<td>-</td>
</tr>
<tr>
<td>(2,1)</td>
<td>-</td>
<td>Y/N?</td>
</tr>
<tr>
<td>(1,3)</td>
<td>Y/N?</td>
<td>-</td>
</tr>
<tr>
<td>(3,1)</td>
<td>-</td>
<td>Y/N?</td>
</tr>
</tbody>
</table>
1) Sketching Logical Topology
1) Sketching Logical Topology

Allow users to select/deselect links between nodes.
1) Sketching Logical Topology

Allow users to select/deselect links between nodes

Sketch-1: \([0,1,2,3,4,5,6,7] \rightarrow [0,1,2,3,4,5,6,7]\)

2-node Azure NDv2, AlltoAll
1) Sketching Logical Topology

Allow users to select/deselect links between nodes

Sketch-1: [0,1,2,3,4,5,6,7] → [0,1,2,3,4,5,6,7]
Sketch-2: 0 → 1

2-node Azure NDv2, AlltoAll
1) Sketching Logical Topology

Allow users to select/deselect links between nodes

Sketch-1: [0,1,2,3,4,5,6,7] → [0,1,2,3,4,5,6,7]
Sketch-2: 0 → 1

Reduces number of links to make decisions about
Can extract high performance over a range of input sizes

2-node Azure NDv2, AlltoAll
2) Sketching NVSwitch connections

Allow users to maximize or minimize unique connections over NVSwitches

Data size: 1.5KB

Data size: 384MB

Throughput (GB/s)

Higher with more connections

Lower with more connections

Number of connections (n)
2) Sketching NVSwitch connections

Allow users to maximize or minimize unique connections over NVSwitches

Data size: 1.5KB

Data size: 384MB

Used to guide algorithm synthesis to be performant for a particular range of input sizes
3) Sketching for symmetry

- Designer can annotate symmetry planes around which data transfers will be fixed to be rotationally symmetric

Reduces number of transfers to make decisions about
Generating the algorithm
Stage-wise synthesis simplifies the problem!
TACCL Synthesizer

Routing
Stage 1

Ordering
Stage 2

Batching
Stage 3

Example collective

Pre-condition

Post-condition
Determine the path that each data chunk will take
(Ordering between data chunks not decided yet)
Solve an ILP to obtain optimal paths (based on congestion and dilation metrics)
Order chunks sent over the same link
Ordering is done using heuristics, e.g., by giving more preference to chunks that need to travel longer distance
TACCL Synthesizer

Routing → Ordering → Batching

Stage 1

Stage 2

Stage 3

Batches data chunks to send them together over links in order to reduce link latency costs.

Solves an ILP to optimize between reduced latency cost v/s possible pipelining gaps.

Example collective

Pre-condition

Post-condition
TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches

(Extends NCCL to implement synthesized collectives)
Evaluation

• Compare performance against NCCL (v2.8.4)
• Collectives: AllGather, AllReduce, AlltoAll
• Topologies:
  • 2-node NVIDIA DGX-2 (32 GPUs)
  • 2-node & 4-node Azure NDv2 (16 GPUs & 32 GPUs)
• Distributed ML models:
  • Transformer-XL, BERT (PyTorch implementation)
How do TACCL algorithms perform against NCCL?

**AllGather**

<table>
<thead>
<tr>
<th>Buffer Size</th>
<th>TACCL</th>
<th>NCCL</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>6.2x</td>
<td>1KB</td>
<td>6.2x</td>
</tr>
<tr>
<td>16KB</td>
<td>5.7x</td>
<td>16KB</td>
<td>5.7x</td>
</tr>
<tr>
<td>256KB</td>
<td>5.1x</td>
<td>256KB</td>
<td>5.1x</td>
</tr>
<tr>
<td>4MB</td>
<td>3.2x</td>
<td>4MB</td>
<td>3.2x</td>
</tr>
<tr>
<td>64MB</td>
<td>1.1x</td>
<td>64MB</td>
<td>1.1x</td>
</tr>
<tr>
<td>1GB</td>
<td>1.25x</td>
<td>1GB</td>
<td>1.25x</td>
</tr>
</tbody>
</table>

**AllReduce**

<table>
<thead>
<tr>
<th>Buffer Size</th>
<th>TACCL</th>
<th>NCCL</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1KB</td>
<td>3.1x</td>
<td>1KB</td>
<td>3.1x</td>
</tr>
<tr>
<td>16KB</td>
<td>6.4x</td>
<td>16KB</td>
<td>6.4x</td>
</tr>
<tr>
<td>256KB</td>
<td>4.1x</td>
<td>256KB</td>
<td>4.1x</td>
</tr>
<tr>
<td>4MB</td>
<td>1.9x</td>
<td>4MB</td>
<td>1.9x</td>
</tr>
<tr>
<td>64MB</td>
<td>1.0x</td>
<td>64MB</td>
<td>1.0x</td>
</tr>
<tr>
<td>1GB</td>
<td>0.95x</td>
<td>1GB</td>
<td>0.95x</td>
</tr>
</tbody>
</table>

Higher is better
How do TACCL algorithms perform against NCCL?

Algorithms synthesized by TACCL are faster than NCCL over a range of input sizes for the evaluated collectives and topologies.
Do we see speedups in end-to-end model training?

**Transformer- XL (Data Parallelism)**

<table>
<thead>
<tr>
<th>Batch size</th>
<th>NCCL</th>
<th>TACCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2x</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1.8x</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1.4x</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>1.1x</td>
<td></td>
</tr>
</tbody>
</table>

**BERT (Model Parallelism)**

<table>
<thead>
<tr>
<th>Batch size</th>
<th>NCCL</th>
<th>TACCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.1x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.6x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1.4x</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>2x</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2.4x</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>1.4x</td>
<td></td>
</tr>
</tbody>
</table>

Microsoft-internal Mixture-of-Experts workload: 17% speedup
Do we see speedups in end-to-end model training?

Transformer- XL (Data Parallelism) vs. BERT (Model Parallelism)

- Transformer- XL: Throughput (seq/s) 100
- BERT: Throughput (tok/s) 60

Speeding up the collective algorithm speeds up end-to-end model training

Microsoft-internal Mixture-of-Experts workload: 17% speedup
Conclusion

TACCL is a tool to synthesize efficient algorithms for collectives
• Guided using intuitive communication sketches
• Solved using novel 3-stage synthesizer

Will soon be available at https://github.com/microsoft/taccl