Distributed Inference: Motivation and Goal

- Support large-model inference using multi-gpu

- Specifically: enable ONNX representation of a model optimized for a multi-processor target
  - Enable a parallelization-optimizer to be expressed as an ONNX-to-ONNX transformer
  - The input to such an optimizer (a model without explicit parallel ops/constructs) can already be expressed in ONNX

Design Choices and Issues

- Communication ops: higher-level ops vs. lower-level primitives:
  - Collective Communication Ops (like NCCL) vs. send/receive primitives
- Parallelism ops: how to represent parallelism?
  - A collection of models, one per target-processor (rank)
  - A single model, with *new parallel-ops* like *parallel-for*
- Representing external-state-dependence (of communication ops)
- Representing processor-id (type)

Communication Ops

- Collective Communication Ops (such as AllReduce, AllGather, Reduce, ReduceScatter, Broadcast) are examples of higher-level ops

- Lower-level primitives: send/receive/broadcast as primitive communication ops

- Proposal:
  - Support both
  - Express higher-level ops as functions over lower-level primitives
  - Same reasoning as for existing approach for tensor-ops

Example function definition

```
def RingReduceSum (rank, total_ranks, input):
    succ = (rank + 1) % total_ranks
    pred = (rank - 1) % total_ranks
    next = input[rank]
    for i in range(total_ranks):
        Send(next, to = succ )
        next_received = Receive(from = pred)
        next_local = input [(rank-i-1) % total_ranks]
        next = Add (next_received, next_local)
    for i in range(total_ranks):
        output[(rank - i) % total_ranks] = next
        Send(next, to = succ)
        next = Receive(from = pred)
```

Parallelism Ops

- How to represent the model?
  - A collection of models, one per target-processor (rank) … no need for parallelism ops
    - Could be same model for all ranks, or potentially different models for different ranks
  - A single model, with new parallelism-ops like *parallel-for*
    - Return-value of parallel-for: e.g., return value computed by processor 0

```
def single_rank (rank, total_ranks, input):
    output = RingReduceSum (rank, total_ranks, input)
```

Model for each rank (processor)

```
def whole_program(total_ranks, input):
    def single_rank_graph (rank):
        return RingReduceSum (rank, total_ranks, input)
    output = ParallelFor (total_ranks,
                          body=single_rank_graph)
```

Complete Model

Communication ops are not pure functions

- Communication ops cannot be reordered, unlike pure ops
- Capturing side-effects of ops (like send/receive)
  - Via dummy inputs/outputs, or
  - Adding an annotation to the op-schema of an op (so topological-sort can take this into account)

Processor-id

- Typing: As an integer (handle) vs. an abstract/opaque type

- Using integer-type simplifies encoding ring-based algorithms (sending and receiving from next/previous processor, etc.)

```
def RingReduceSum (rank, total_ranks, input):
    succ = (rank + 1) % total_ranks
    pred = (rank - 1) % total_ranks
    …
```

- Explicit (as parameter) vs. implicit