Implementing Single-Dim Strategies with Strategy Validator

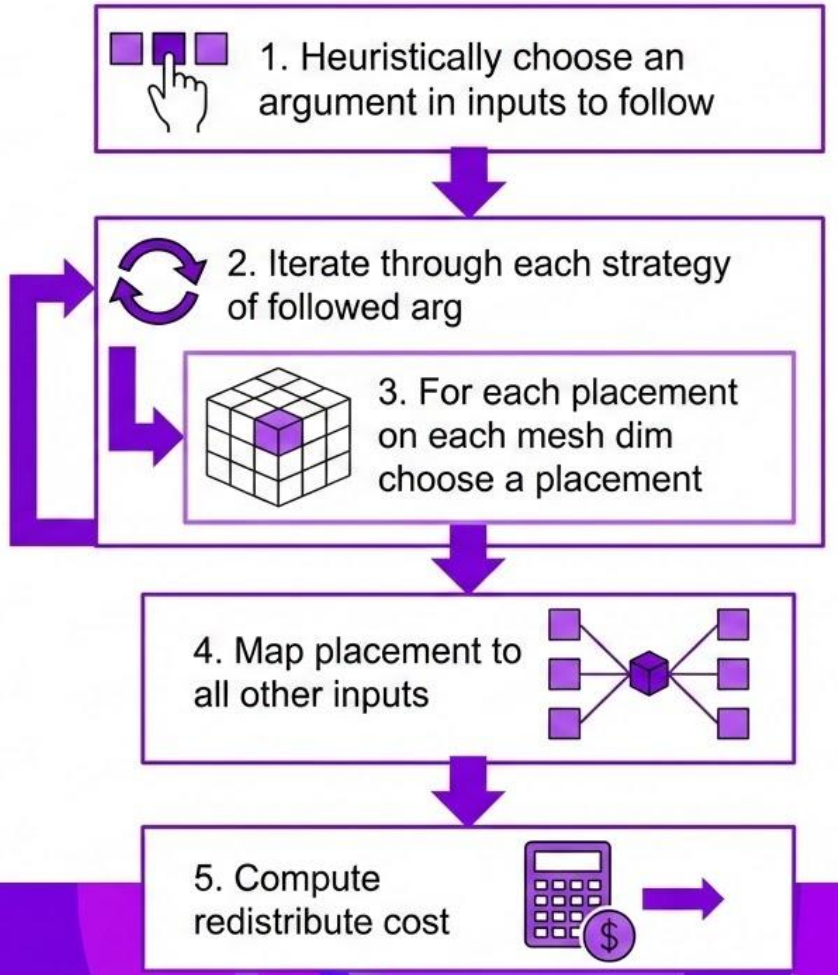
Anshul Sinha, PyTorch @ Meta MSL

PyTorch CONFERENCE EUROPE 2026

Old Path: Full Op Strategy

Tries to do too much in one function

- Enumerate N-D mesh strategies
- Filter out bad strategies
- follow-the-leader approach (premature optimization)
- Compute redistribute cost



Problems

1. Difficult to add rules quickly
2. Hard to prove you're not missing prop rules
3. Difficult to review correctness

Result: Error Prone Strategies

Example

torch.max

Rank 0:
DT_0: a, p(max)
DT_1: c, p(max)

Rank 1:
DT_0: b, p(max)
DT_1: d, p(max)

Redistribute
Rank 0:
DT_0: max(a, b), R
DT_1: max(c, d), R

max(max(a, b), max(c, d))

Propagate then Redistribute
Rank 0:
DT_0: max(a, c), p(max)
DT_1: max(b, d), p(max)

Redistribute
max(max(a, c), max(b, d))

```
elif isinstance(placement, Partial):
    # Check if this partial type should be preserved
    if preserve_partial is not None and placement.is_partial(
        preserve_partial
    ):
        out_placements.append(placement)
    # note that only partial-sum and partial-avg are supported for linearity
    elif linearity >= 0 and (
        placement.is_partial("sum") or placement.is_partial("avg")
    ):
        # propagate the partial placement
        out_placements.append(placement)
    else:
        # clear the partial placement if op does not support linearity
        # by default we just replicate the partial, need to see if this
        # is optimal for all cases
        out_placements.append(Replicate())
```

New Rule

New Branch

Single Dim Strategies

- Takes advantage that each mesh dim is independent
- Separates two concerns combined in OpStrategy
  - semantic correctness—valid input/output sharding combinations for an operator
  - search-space pruning to avoid combinatorial blowups on N-dimensional meshes

- Op provides valid input/output sharding rules for a single dimension
- Single dim infra takes care of expanding across full mesh, selects lowest cost strategy

Pointwise Single Dim Strategies Example

- Infra for common strategies across all pointwise_ops
- For sub-category of pointwise_ops, add extra rules specific to them
- → Follow this rule for all kinds of ops

Extremely easy to add and review!

```python
# Binary ops monotonically increasing in both arguments.
# max-preserving: P(max)+P(max)->P(max) because max(max(a),max(b)) = max(a,b)
monotonic_max_preserving_binary_ops: list[OpOverload] = [
    aten.clamp_min.Tensor,
    aten.fmax.default,
    aten.fmax.out,
    aten.maximum.default,
    aten.maximum.out,
    prims.fmax.default,
    # foreach variants
    aten._foreach_maximum_.List,
]

_MONOTONE_MAX_PRESERVING_BINARY_BASE_RULES: list[list[Placement]] = [
    *_MONOTONE_BINARY_BASE_RULES,
    [Partial("max"), Partial("max"), Partial("max")],
]

for op in monotonic_max_preserving_binary_ops:
    _register_single_dim_pointwise(op, _MONOTONE_MAX_PRESERVING_BINARY_BASE_RULES)
```

Composing Across a Multi-Dim Mesh

```
maximum single-dim rules:
  [S(0), S(0), S(0)]           # shard both inputs + output on same dim
  [P(max), P(max), R]          # partial-max in a, replicate b
  [P(max), R, P(max)]          # replicate a, partial-max in b
  [P(max), P(max), P(max)]     # both partial-max -> partial-max
  [P(min), P(min), R]          # partial-min monotonic
  [P(min), R, P(min)]          # partial-min monotonic

Expand on 2D mesh [a, b]:

dim a rule              dim b rule              output          input1          input2
──────────              ──────────              ──────          ──────          ──────
S(0)->S(0),S(0)     x   S(0)->S(0),S(0)     =   (S(0),S(0))     (S(0),S(0))     (S(0),S(0))
S(0)->S(0),S(0)     x   P(max)->P(max),R    =   (S(0),P(max))   (S(0),P(max))   (S(0),R)
P(max)->P(max),R    x   S(0)->S(0),S(0)     =   (P(max),S(0))   (P(max),S(0))   (R,S(0))
P(max)->P(max),R    x   P(max)->P(max),R    =   (P(max),P(max)) (P(max),P(max)) (R,R)
P(max)->P(max),P(max) x P(min)->P(min),R    =   ✗ mixed partial (max+min on same tensor)
...etc
```

Strategy Validator

- Run it before migrating ops to find missing rules
- Run it after finalizing rules for set of ops to verify correctness

Strategy Validation Example

python -m torch.distributed.tensor._ops.strategy_validation --op argmax

- Detects incorrectly implemented rules

```
Comparing operator: argmax

================================================================================
Found 1 OpInfo(s) for 'argmax'
World size: 2
    Processing 18 sample inputs...

================================================================================
COMPARISON SUMMARY

================================================================================
Total samples processed: 16
Total combinations tested: 384
Elapsed time: 1.87s

True positives (both agree valid): 32
DTensor incorrect: 3 rules over 12 samples
DTensor missing: 0

--- DTENSOR INCORRECT (has rule but ground truth invalid) ---

  [aten.argmax.default]
    P(max) -> P(max)
      Sample 0: [[4, 3]]
      Sample 3: [[8, 4]], {'dim': 0}
      Sample 5: [[4, 3, 2]], {'dim': 1}
      ... and 9 more
```

How to Contribute

- Add ops that aren't currently covered by DTensor

- Add single dim strategies for your custom ops
  - Create OpInfos
  - Use Strategy Validator to find new rules
  - Implement strategies
  - Use Strategy Validator to verify correctness

https://github.com/pytorch/pytorch/tree/main/torch/distributed/tensor/_ops