Automatic Parallelization in PyTorch

Francisco Massa[1], Will Constable[1], Brian Hirsh[1], Edward Yang[1], Ivan Kobzarev[1], Elias Ellison[1], Pei Zhang[1], Sanket Purandare[1], Simon Fan[1], Ruisi Zhang[1], Milad Mohammadi[1], Davide Italiano[1], Peng Wu[1], Patrick Labatut[1]

[1]Meta

PyTorch CONFERENCE 2025

Problem and Motivation

- Efficiently scaling distributed training of large models on GPU clusters requires combining multiple parallelization strategies (e.g., Data, Model, Operator).
- Selecting the optimal strategy mix for a specific model, batch size, and cluster setup relies heavily on expert intuition and trial and error.
- Implementing the chosen strategy remains a manual, tedious process and often involves baking a parallelization strategy in model code.

DTensor and DeviceMesh: Core abstractions for specifying and executing an SPMD strategy

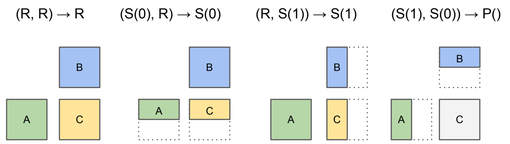


DTensor Placements:

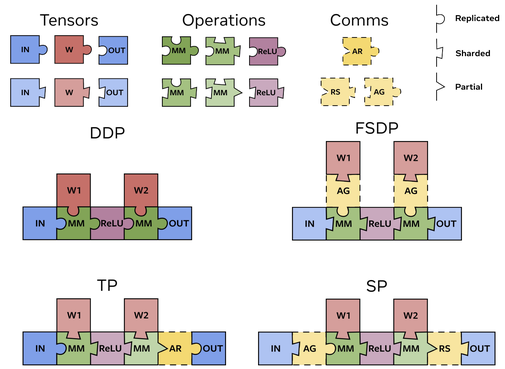
- **Shard (S)**: shard on a tensor dimension across device mesh dimension
- **Replicate (R)**: replicate tensor across device mesh dimension
- **Partial (P)**: describes a tensor pending reduction across a device mesh dimension

DeviceMesh: An abstraction that describes the globalview/topology of devices within a cluster i.e. 2-D mesh [[0,1], [2, 3]]

Example Matmul: C = A @ B



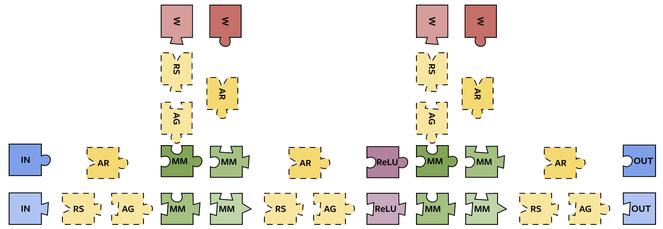
Key Insight: Finding an optimal parallelization strategy is like solving a jigsaw puzzle of DTensor Placements



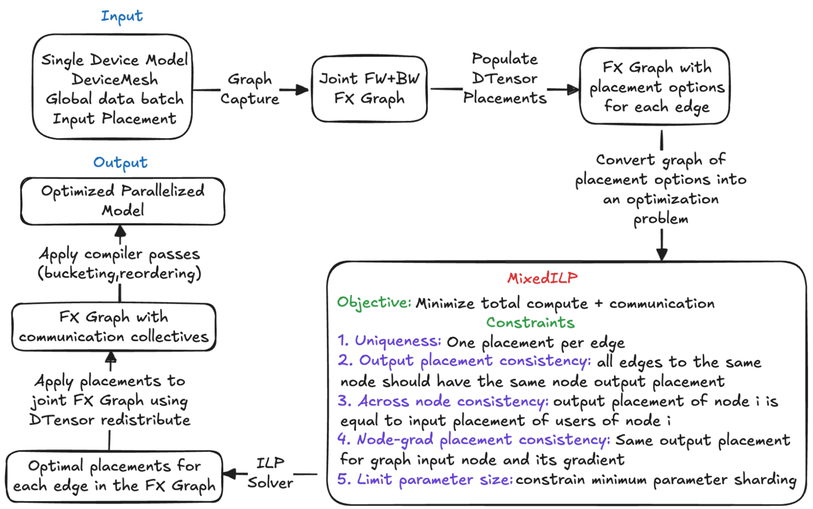
DDP/FSDP/TP are just different combinations of DTensor Placements for each operator's inputs and ouputs

AutoParallel: Optimizing model sharding with global constraints

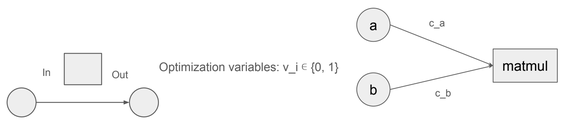
It simplifies parallelism by allowing a "system" to find an "optimal" **SPMD** solution by finding the cost-effective pieces that complete the jigsaw puzzle



AutoParallel: System Design



Examples



Input placements

Output placements	R	S(0)	S(1)
R	v_0	v_1	v_2
S(0)	v_3	v_4	v_5
S(1)	v_6	v_7	v_8

$\sum(v_i) = 1$

Input **a** placements

c_a	R	S(0)	S(1)
(R, R)→R	a_0	a_1	a_2
(R, S(1))→S(1)	a_3	a_4	a_5
(S(0), R)→S(0)	a_6	a_7	a_8

Input **b** placements

c_b	R	S(0)	S(1)
(R, R)→R	b_0	b_1	b_2
(R, S(1))→S(1)	b_3	b_4	b_5
(S(0), R)→S(0)	b_6	b_7	b_8

a_0 + a_1 + a_2 = b_0 + b_1 + b_2
a_3 + a_4 + a_5 = b_3 + b_4 + b_5
a_6 + a_7 + a_8 = b_6 + b_7 + b_8

Results

Llama 3 8B Configs	TPS (% relative to baseline)	Peak Memory (GiB)
1D DeviceMesh [64] with 64 GPUs		
FSDP+compile (baseline)	5973	24.4
AutoParallel+1D+compile	4608 (77%)	22.1 (9% improved)
AutoParallel+1D+compile+ bucketing+reordering	**5819 (97%)**	**25.1 (2.8% worse)**
2D DeviceMesh [8, 8] with 64 GPUs		
FSDP+TP+compile (baseline)	3900	9.23
AutoParallel+2D+compile	2579 (66%)	10.6 (15% worse)
AutoParallel+2D+compile+ bucketing+reordering	**3468 (89%)**	**10.9 (18% worse)**

The experiments are conducted on NVIDIA H100 GPUs with 95 GiB memory, where each host is equipped with 8 GPUs and NVSwitch. Two hosts form a rack connected to a TOR switch. A backend RDMA network connects the TOR switches.

Ongoing Work

- Support for local_map to apply a custom function to the local shards of a DTensor.
- Support for Expert Parallelism with token-choice routing using custom DTensor placements and local_map.
- Support for Context Parallelism (Ring Attention) and Pipeline Parallelism (with advanced schedules like Zero Bubble and DualPipeV).

Github

AutoParallel Github

https://github.com/meta-pytorch/autoparallel