# PR #2031 完整报告

radixark/miles

feat(fsdp): add hybrid sharding

合并时间：2026-08-04 03:45

原文链接：http://prhub.com.cn/radixark/miles/pull/2031

## 执行摘要

- 一句话：FSDP 后端新增混合分片，配梯度一致性测试
- 推荐动作：值得精读。重点学习 build_fsdp_meshes 的 _unflatten 用法与 " 逻辑 DP 域不变、仅 FSDP 分片域切分 " 的设计取舍，以及梯度 parity worker 的三层断言方法（参考梯度对比、跨 rank bit-exact、权重还原）。建议关注 zhihengy 后续的 FSDP + CP 支持 PR，以及 _unflatten 在 PyTorch 升级后的兼容性。

## 功能与动机

PR body 明确说明：FSDP 此前始终使用一维全分片 mesh（"FSDP previously always used a one-dimensional full-shard mesh"），参数分片组即整个 DP 域，通信粒度不可调。Hybrid sharding 的目标是 "reduces each parameter-shard group while retaining replicas across groups, without changing the logical data-parallel domain used for data loading and metric reduction"——即在不改变数据并行语义的前提下，把分片组缩小、跨组保留复制，从而在大规模训练中摊薄 all-gather/reduce-scatter 的通信压力。CP 支持被显式声明为 out of scope。

## 实现拆解

实现按 5 步展开：

1. 参数与启动期校验：miles/backends/experimental/fsdp_utils/arguments.py 在 FSDPArgs 数据类新增 dp_replicate_size: int = 1 字段（默认 1，保证完全向后兼容），并新增 validate_hybrid_shard_args 做三层校验：dp_replicate_size >= 1、world_size % context_parallel_size == 0、data_parallel_size % dp_replicate_size == 0，确保后续 _unflatten 能构成完整方形 mesh。miles/utils/arguments.py 在 parse_args 的非 pipeline 校验分支调用该校验，使非法拓扑在启动期即被拦截。
2. mesh 构造核心：miles/backends/experimental/fsdp_utils/parallel.py 新增 build_fsdp_meshes，先用 init_device_mesh 建立 (dp, cp) 二维网格，再在 dp_replicate_size > 1 时对 dp_mesh 执行 _unflatten(0, (dp_replicate_size, dp_shard_size), ("dp_replicate", "dp_shard")) 得到 FSDP 专用二维 mesh；函数返回 dp_cp、dp、cp、fsdp 四个命名视图。create_fsdp_parallel_state 改为消费该函数，ParallelState 的 intra_dp/cp 等 GroupInfo 从显式 mesh 视图取 group。

3. 共享接口与消费方改造：miles/backends/training_utils/parallel.py 为 ParallelState 新增 meshes: dict[str, DeviceMesh] 字段与 get_mesh(name) 方法，作为后端 mesh 视图的共享出口；miles/backends/experimental/fsdp_utils/actor.py 中主模型与 ref 模型的 apply_fsdp2、_fsdp2_load_full_state_dict 全部从 get_mesh("fsdp") 取 mesh，取代原 dp_mesh 属性，确保初始化与权重同步走混合分片拓扑。
4. 确定性梯度测试：tests/fast-gpu/_fsdp_hybrid_shard_worker.py 实现 4 rank worker，以三种方式断言混合分片正确性——materialize 后的 DTensor 梯度与全量 all-reduce 参考梯度一致（容差 2e-6 量级）、跨 rank 全收集逐对 bit-exact 比较、gather_full_param 还原初始权重；tests/fast-gpu/test_fsdp_hybrid_shard.py 通过 torch.distributed.run 依次跑 r1s4/r2s2/r4s1 并做拓扑间交叉比对。
5. e2e CI 与文档配套：reviewer 在最后提交补齐 tests/e2e/fsdp/test_qwen3_4B_fsdp_hybrid_shard_r2s2.py（4 GPU H200）与 r2s4.py（8 GPU H100）两个 GRPO 端到端 RL 用例并注册到 CI suite；tests/fast/backends/test_fsdp_qwen3_true_on_policy.py 的 monkeypatch 从 dp_mesh 属性改为 get_mesh 方法以对齐新接口；docs/user-guide/cli-reference.md 登记 --dp-replicate-size 参数说明。

关键文件：

- miles/backends/experimental/fsdp_utils/parallel.py（模块 并行构造；类别 source；类型 core-logic；符号 build_fsdp_meshes, create_fsdp_parallel_state）：核心实现文件：新增 build_fsdp_meshes 构造 DP/CP/FSDP 命名 mesh 视图，并用 DeviceMesh._unflatten 把 dp 维度拆成 (dp_replicate, dp_shard)，是 hybrid sharding 的基石。
- miles/backends/experimental/fsdp_utils/arguments.py（模块 参数校验；类别 source；类型 core-logic；符号 dp_replicate_size, validate_hybrid_shard_args）：新增 dp_replicate_size 参数定义与 validate_hybrid_shard_args 拓扑校验，保证非法拓扑在启动期被拦截。
- tests/fast-gpu/_fsdp_hybrid_shard_worker.py（模块 梯度校验；类别 test；类型 test-coverage；符号 _Block, _TinyModel, _make_model, _materialize_gradient）：4 rank 梯度一致性 worker，定义了三层断言方法（参考梯度、跨 rank bit-exact、权重还原），是验证 hybrid 拓扑正确性的核心测试资产。
- tests/fast-gpu/test_fsdp_hybrid_shard.py（模块 梯度测试；类别 test；类型 test-coverage；符号 _run_worker, test_fsdp_hybrid_shard_gradient_parity）：以 subprocess 拉起 4 卡 worker 依次跑 r1s4/r2s2/r4s1 并做拓扑间梯度交叉比对，注册到 stage-c-4-gpu-h200 CI。
- miles/backends/experimental/fsdp_utils/actor.py（模块 训练执行；类别 source；类型 core-logic；符号 init, _create_ref_model）：FSDP 训练执行入口：apply_fsdp2 与 state-dict 加载从 dp_mesh 切换到 get_mesh("fsdp")，是混合分片真正生效的消费方。
- miles/backends/training_utils/parallel.py（模块 并行状态；类别 source；类型 core-logic；符号 ParallelState, get_mesh）：共享并行状态扩展：ParallelState 新增 meshes 字典与 get_mesh 方法，是 mesh 视图跨模块流通的公共接口。
- tests/e2e/fsdp/test_qwen3_4B_fsdp_hybrid_shard_r2s2.py（模块 端到端 CI；类别 test；类型 test-coverage；符号 prepare, execute）：reviewer 补的端到端 RL CI：

Qwen3-4B 在 4 GPU H200 上以 --dp-replicate-size 2 跑 GRPO，覆盖 r2s2 拓扑。

- tests/e2e/fsdp/test_qwen3_4B_fsdp_hybrid_shard_r2s4.py（模块 端到端 CI；类别 test；类型 test-coverage；符号 prepare, execute）：reviewer 补的端到端 RL CI：Qwen3-4B 在 8 GPU H100 上以 --dp-replicate-size 2 跑 GRPO，覆盖 r2s4 拓扑。
- miles/utils/arguments.py（模块 参数入口；类别 source；类型 dependency-wiring；符号 parse_args）：通用参数解析入口：在非 pipeline 校验分支调用 validate_hybrid_shard_args，让 FSDP 拓扑校验对 CLI 生效。
- tests/fast/backends/test_fsdp_qwen3_true_on_policy.py（模块 单测适配；类别 test；类型 test-coverage）：现有 FSDP 单测适配新接口：monkeypatch 从 dp_mesh 属性改为 get_mesh 方法，确保接口迁移不破坏既有测试。
- docs/user-guide/cli-reference.md（模块 文档；类别 docs；类型 documentation）：CLI 参考文档登记 --dp-replicate-size，保证公共参数有文档可查。

关键符号：build_fsdp_meshes, create_fsdp_parallel_state, validate_hybrid_shard_args, get_mesh, test_fsdp_hybrid_shard_gradient_parity, _run_worker, _fully_shard_model, _materialize_gradient, _reference_gradients

## 关键源码片段

### miles/backends/experimental/fsdp_utils/arguments.py

新增 dp_replicate_size 参数定义与 validate_hybrid_shard_args 拓扑校验，保证非法拓扑在启动期被拦截。

```
# miles/backends/experimental/fsdp_utils/arguments.py

def validate_hybrid_shard_args(args) -> None:
    """校验训练拓扑能否构成请求的 FSDP2 hybrid-shard mesh。"""
    replicate_size = args.dp_replicate_size
    if replicate_size < 1:
        raise ValueError(f"dp_replicate_size must be at least 1, got {replicate_size}")

    world_size = args.actor_num_nodes * args.actor_num_gpus_per_node
    if args.context_parallel_size < 1:
        raise ValueError(
            f"context_parallel_size must be at least 1, got {args.context_parallel_size}"
        )
    if world_size % args.context_parallel_size:
        raise ValueError(
            f"world_size({world_size}) must be divisible by "
            f"context_parallel_size({args.context_parallel_size})"
        )

    # 只有 data_parallel_size 能被复制组大小整除时，
    # _unflatten((dp_replicate, dp_shard)) 才能构成完整方形 mesh
    data_parallel_size = world_size // args.context_parallel_size
    if data_parallel_size % replicate_size:
        raise ValueError(
```

```
            f"data_parallel_size({data_parallel_size}) must be divisible by "
            f"dp_replicate_size({replicate_size})"
        )
```

## tests/fast-gpu/_fsdp_hybrid_shard_worker.py

4 rank 梯度一致性 worker，定义了三层断言方法（参考梯度、跨 rank bit-exact、权重还原），是验证 hybrid 拓扑正确性的核心测试资产。

```
# tests/fast-gpu/_fsdp_hybrid_shard_worker.py
# 4 卡 worker：对 r1s4、r2s2、r4s1 三种拓扑做 FSDP2 梯度一致性校验

def _materialize_gradient(param: nn.Parameter) -> torch.Tensor:
    # FSDP2 的梯度是 DTensor（按分片组切分），先 materialize 回完整张量再比较
    grad = param.grad
    assert grad is not None
    return grad.full_tensor() if isinstance(grad, DTensor) else grad


def _reference_gradients(
    inputs: torch.Tensor, world_size: int
) -> dict[str, torch.Tensor]:
    # 参考梯度：未分片模型直接反向，再把梯度跨全部 rank 做 all-reduce 取平均，
    # 得到与数据并行语义一致的期望梯度
    model = _make_model()
    model(inputs).square().mean().backward()

    gradients = {}
    for name, param in model.named_parameters():
        assert param.grad is not None
        gradient = param.grad.detach().clone()
        dist.all_reduce(gradient)
        gradient /= world_size
        gradients[name] = gradient
    return gradients


def _fully_shard_model(model: _TinyModel, mesh) -> None:
    # 每个 Block 一层 FSDP2 嵌套分片，最后整个模型再包一层，
    # 与真实训练中逐层 fully_shard 的用法一致
    for block in model.blocks:
        fully_shard(block, mesh=mesh)
    fully_shard(model, mesh=mesh)


def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument("--replicate-size", type=int, required=True)
    parser.add_argument("--output", required=True)
    args = parser.parse_args()
```

```
local_rank = int(os.environ["LOCAL_RANK"])
torch.cuda.set_device(local_rank)
dist.init_process_group("nccl", device_id=torch.device("cuda", local_rank))
rank = dist.get_rank()
world_size = dist.get_world_size()
shard_size = world_size // args.replicate_size

meshes = build_fsdp_meshes(
    device_type="cuda",
    world_size=world_size,
    context_parallel_size=1,
    dp_replicate_size=args.replicate_size,
)
fsdp_mesh = meshes["fsdp"]
# 单复制组时退化为 1D mesh，多复制组时为 2D (dp_replicate, dp_shard)
assert fsdp_mesh.ndim == (1 if args.replicate_size == 1 else 2)

generator = torch.Generator(device="cuda").manual_seed(9000 + rank)
inputs = torch.randn(8, 32, generator=generator, device="cuda")
expected_gradients = _reference_gradients(inputs, world_size)

model = _make_model()
initial_weights = {
    name: param.detach().clone() for name, param in model.named_parameters()
}
_fully_shard_model(model, fsdp_mesh)
model(inputs).square().mean().backward()

gradients = {}
for name, param in model.named_parameters():
    actual = _materialize_gradient(param).detach()
    # 1) 与全量 all-reduce 参考梯度对比，误差控制在 2e-6 量级
    torch.testing.assert_close(actual, expected_gradients[name], rtol=2e-5, atol=2e-6)

    # 2) 跨 rank 全收集后逐对比较：混合分片下每个复制组计算出的
    # 梯度必须完全一致（bit-exact），否则 on-policy 语义会被破坏
    peers = [torch.empty_like(actual) for _ in range(world_size)]
    dist.all_gather(peers, actual)
    for peer in peers[1:]:
        torch.testing.assert_close(peers[0], peer, rtol=0, atol=0)
    gradients[name] = actual.cpu()

    # 3) 分片参数经 gather_full_param 必须还原为初始权重
    full_param = gather_full_param(param)
    torch.testing.assert_close(full_param, initial_weights[name])

if rank == 0:
    torch.save(gradients, args.output)
    print(f"PASS r{args.replicate_size}s{shard_size}", flush=True)
```

```
    dist.barrier()
    dist.destroy_process_group()
```

## 评论区精华

核心讨论围绕验证边界展开：

- CP 覆盖问题：Zhichenzzz 在 issue 评论中问 "Great feature! btw is cp testing covered in this PR?"；zhihengy 回复 LGTM，并说明与作者在 Miles-diffusion 实现 HSDP 时已讨论过方案，CP 问题正在单独排查，会另开 PR 附实验与测试。结论是 CP 明确排除在本 PR 外，但 mesh 构造已预留 cp 视图。
- 验证证据边界：PR body 将 Qwen3.5-4B 的 exploratory 对比（train/rollout log-prob 差 0.02516 vs 0.01817）明确标注为 "retained for follow-up but is not used as validation evidence"，只以确定性梯度测试和 Qwen3-4B 受控 RL 对照作为验收证据，体现了严格的验证纪律。
- Reviewer 追加 CI：Zhichenzzz review APPROVED 并承诺 "I will add one more hybrid dp ci e2e test"，最终以提交 dfeaf8b（author 为 Zhichenzzz）落地 r2s2 与 r2s4 两个 e2e 用例，形成 author 实现 + reviewer 补测的协作模式。
  - CP 测试覆盖是否包含在 PR 内 (question): CP 明确排除在本 PR 范围之外，由后续独立 PR 跟进；本 PR 的 mesh 构造已预留 cp 维度视图。
  - HSDP 方案与 Miles-diffusion 实现对齐 (design): 方案获得资深 reviewer 背书，无新增设计疑虑。
  - 补充 hybrid DP 端到端 CI 测试 (testing): 已落地，e2e 覆盖两种多卡拓扑。

## 风险与影响

- 风险：技术风险集中在四方面：
- 依赖半公开 API：build_fsdp_meshes 使用 DeviceMesh._unflatten（下划线前缀的半公开 API），PyTorch 版本升级可能变更签名或行为，需要版本锁定或封装。
- 共享接口改动面：ParallelState.meshes/get_mesh 面向所有 FSDP 消费方开放，若有其它模块仍直接访问 dp_mesh 属性会静默回到旧拓扑；本 PR 仅改了 actor.py 两处调用与对应单测，后续新增消费方需警惕。
- CP 组合盲区：校验逻辑允许 context_parallel_size > 1 与 hybrid 同时启用，但所有测试仅覆盖 cp_size=1，FSDP + CP + hybrid 三者的交错语义（含 ring_flash_attn 与 _unflatten）未验证。
- 验证覆盖缺口：PR checklist 显示 pre-commit、全量 pytest -x、train.py --help 均未运行；e2e RL 是 100 rollout 小规模对照，非统计显著。此外 Qwen3.5-4B exploratory 数据中 hybrid 的 train/rollout log-prob 差略高于 baseline，提示部分模型上 HSDP 数值路径仍可能有细微偏差。
- 影响：影响范围评估：
- 用户侧：FSDP 用户新增 --dp-replicate-size 开关，默认 1 时行为与原一维全分片完全一致，向后兼容；大规模训练可通过复制组摊薄分片通信，为跨节点 / 跨机柜拓扑调优提供新自由度。

- 系统侧：FSDP 后端 mesh 语义从单一 dp_mesh 升级为命名 mesh 视图集合，ParallelState 成为 mesh 共享出口，后续 CP 支持可直接复用 meshes 字典，属于并行基础层的结构性扩展。
- 团队侧：确立了 " 确定性梯度 parity + 受控 RL 对照 + e2e CI" 三层验证范式，并已固化进 CI 体系（stage-c-4-gpu-h200 / stage-c-8-gpu-h100）；worker 的三种断言方式可作为其它并行策略的测试模板。
- 风险标记：依赖半公开 API _unflatten，CP 组合未覆盖，完整测试套件未全量运行，并行状态接口面向所有 FSDP 路径开放

## 关联脉络

- PR #2045 codeowners: cover miles/backends/experimental/fsdp_utils: 同一目录 miles/backends/experimental/fsdp_utils 的 CODEOWNERS 覆盖，本 PR 改动了该目录下 parallel.py、arguments.py、actor.py，所有者规则有助于后续 FSDP 变更的 review 分工。