

PR #7223 完整报告

verl-project/verl

[megatron] feat: PP/VPP support for the steady delta export

合并时间: 2026-08-03 14:50

原文链接: <http://prhub.com.cn/verl-project/verl/pull/7223>

执行摘要

- 一句话: Megatron 稳态 delta 导出新增 PP/VPP 支持
- 推荐动作: 值得精读。这是 delta_sharded 引擎演进的关键一步——它展示了如何不加 relay 就复用 bridge 全局参数目录实现 PP 导出, 其中 `_exchange_slot_tables` 按目录行键控的契约设计、`contributes` 去重位、以及 `placeholder` 行的 zero-count 锁步机制都值得学习。建议重点阅读 `build_export_index` 的三态路由、`_pp_differential` 的独立参照链设计, 以及第二个 commit 对 `ep_size>1` 归错问题的根因分析。

功能与动机

PR body 明确说明这是对 #7060 的后续工作: 此前 Megatron 稳态 delta 导出被 `PP=1` 断言限制, base 版代码的断言信息也写明 "the seed full sync supports PP already, the steady relay is roadmapped"——seed 全量同步早已支持 PP, 但稳态增量同步在 PP 布局下被拦截。核心诉求是让 PP/VPP 布局的 Megatron 训练也能享受 delta_sharded 引擎的稳态增量权重同步, 避免回退到全量 NCCL 同步 (235B 上全量同步需 220.5-236.8s, delta 稳态仅 97.5-111.1s)。

实现拆解

变更入口在 `verl/workers/engine/megatron/delta_export.py` 的 `build_export_index`, 核心思路是骑乘 Megatron-Bridge 已有的全局参数目录, 而非自行枚举。拆解如下:

1. 解除 `PP=1` 断言, 复用 bridge 全局参数目录: `build_export_index` 删掉 `pp_world == 1` 的 `assert`, 改为读取真实 `pp_world`。`get_conversion_tasks` 原本就会跨 `pp rank` `allgather` 参数名、确定性排序、去重 `tied embeddings`, 并对其他 stage 拥有的参数留下 `param_weight=None` 占位——这正是 `PP>1` 锁步对齐的基础。`McoreParamExport.param` 变为 `Optional[torch.Tensor]`, 并新增 `slots` 字段存放全局 slot 表。
2. `PP>1` 路由改为 `WORLD group` 全参合并: 每个参数统一挂到 `dist.group.WORLD`, 因为 wire master (global rank 0) 必然参与每个 `gather`, 无论参数归哪个 stage, 天然不需要 relay 跳转; `owner-stage rank` 贡献自己的分片, `dp/cp` 副本通过 `ShardSpec(contributes=False)` 去重。`param=None` 的占位任务变成 `full_shape=(0,)`、`contributes=False` 的 zero-count 锁步行, 与 `HSDP` 副本既有的锁步机制一致。`PP=1` 分支保持原路由 (`tp` 组 / `etpxep` 联合组 / 复制型 `rank-0` 直读), 行为逐字节不变。

3. 新增 `_exchange_slot_tables` 一次性全局交换：在 index 构建末尾按目录行（而非参数名）键控做 slot table union，owner rank 先探测一次 zero delta 完成预 seed，非 owner rank 从第一次同步就产出尺寸正确的零计数条目。这修复了 235B 验证暴露的 `ep_size>1` 专家归错：此前各 rank 只缓存自己的 ep-local 专家名，违反引擎 batched gather 按 slot 位置合并的契约。
4. 引擎侧配套：transformer_impl.py 的 `_mcore_export_index` 把 `self._delta_slot_cache` 传入 `build_export_index`（顺序调整保证 cache 先于 index 建立）；`get_per_tensor_param_shard` 对 param is None 的行产出空张量 shard，使跨 stage 占位行能流向 delta 引擎的锁步路径。
5. 测试与验证配套：test_mcore_probe_differential.py 新增 `_pp_differential` 生产路径差分 oracle（支持 PP_SIZE/EP_SIZE 环境变量、新增 qwen2_tied tied-embedding 变体），并与 `bridge.export_hf_weights` 这条独立链做 bitwise 对比；同时顺带修复 transformers 5.6.0 将小型 NemotronH checkpoint 序列化为 `backbone.embedding.weight`（单数）而 bridge 映射使用 `backbone.embeddings.weight` 的键名归一问题。

关键文件：

- `verl/workers/engine/megatron/delta_export.py`（模块 导出引擎；类别 source；类型 core-logic；符号 `build_export_index`, `_exchange_slot_tables`）：核心实现：解除 PP=1 断言，新增 PP>1 的 WORLD group 合并、placeholder 锁步行与一次性全局 slot table 交换，179 行变更的主体。
- `tests/special_distributed/test_mcore_probe_differential.py`（模块 差分测试；类别 test；类型 test-coverage；符号 `_pp_differential`）：新增 `_pp_differential` 生产路径差分 oracle，覆盖 PP/EP 组合、目录锁步、逐行 slot 表一致性与 bitwise 对比，并修复 transformers 5.6.0 键名归一。
- `verl/workers/engine/megatron/transformer_impl.py`（模块 引擎实现；类别 source；类型 core-logic；符号 `_mcore_export_index`, `get_per_tensor_param_shard`）：引擎侧配套：索引构建传入 `slot_cache`，`get_per_tensor_param_shard` 对跨 stage 占位行产出空 shard。

关键符号：`build_export_index`, `_exchange_slot_tables`, `_pp_differential`, `_mcore_export_index`, `get_per_tensor_param_shard`

关键源码片段

`tests/special_distributed/test_mcore_probe_differential.py`

新增 `_pp_differential` 生产路径差分 oracle，覆盖 PP/EP 组合、目录锁步、逐行 slot 表一致性与 bitwise 对比，并修复 transformers 5.6.0 键名归一。

```
def _pp_differential(bridge, model, rank: int, world: int) -> None:
    """PP>1 差分 oracle:跑生产导出路径(全局目录 + slot 预 seed + probe 条目),
    把"每个本地元素都变化"的全量 delta 同步结果与 bridge.export_hf_weights
    对比——后者走 bridge 自身的 PP 广播全量导出,构成独立参照链。
    逐 HF 张量 bitwise 比较,中间还做目录锁步与占位行校验。"""
    from verl.workers.engine.megatron.delta_export import build_export_index, mcore_hf_delta_
    entry
```

```

slot_cache: dict = {}
index = build_export_index(bridge, model, slot_cache)

# 1. 目录锁步：所有 rank 行数必须一致 (mcore 专家参数名在 ep 维度
# 合法不同，但 " 行 " 是合并单位，行数必须对齐)
names = [r.megatron_name for r in index]
all_names: list = [None] * world
dist.all_gather_object(all_names, names)
failures = []
if rank == 0 and any(len(n) != len(names) for n in all_names):
    failures.append(f"directory row counts diverge: {[len(n) for n in all_names]}")

# 1b. 逐行 slot table 一致性 :batched gather 按行内 slot 位置合并，
# rank 0 用自己的表命名合并结果，因此每行表必须全局一致
row_tables = [r.slots for r in index]
all_tables: list = [None] * world
dist.all_gather_object(all_tables, row_tables)
if rank == 0:
    for r_i, tbls in enumerate(all_tables):
        bad = [k for k in range(len(row_tables)) if tbls[k] != row_tables[k][:3]]
        if bad:
            failures.append(f"row slot tables diverge on rank {r_i} (rows {bad})")
            break

# 3. 独立参照 :bridge 自带的全量导出 ( 内部走 PP broadcast, 各 rank 全量张量 )
exported = bridge.export_hf_weights(model, cpu=True, show_progress=False)
ref = {name: t.detach().to(torch.bfloat16).cpu() for name, t in exported}

all_pieces: list = [None] * world
dist.all_gather_object(all_pieces, mine)
n_checked = n_pass = 0
if rank == 0:
    slot_names = {k for d in all_pieces for k in d}
    missing_ref = sorted(set(ref) - slot_names)
    if missing_ref:
        more = max(0, len(missing_ref) - 5)
        failures.append(f"HF tensors never covered by any entry: {missing_ref[:5]} (+{more})")
    for sname in sorted(slot_names):
        if sname not in ref:
            failures.append(f"entry slot {sname} absent from bridge export")
            continue
        full = ref[sname]
        merged = torch.full_like(full, float("nan")) # NaN 哨兵：未覆盖位必须是 NaN
        covered = torch.zeros(full.numel(), dtype=torch.int32)
        mflat = merged.view(-1)
        for d in all_pieces:
            got = d.get(sname)
            if got is None:
                continue

```

```

sshape, pos, val = got
if tuple(sshape) != tuple(full.shape):
    failures.append(f"{sname}: slot shape {sshape} != export shape {tuple(full.shape)}")
    continue
both = covered[pos] > 0
if both.any() and not torch.equal(mflat[pos[both]].view(torch.int16), val[both].view(torch.int16)):
    failures.append(f"{sname}: conflicting rank contributions")
mflat[pos] = val
covered[pos] += 1
n_checked += 1
if torch.isnan(mflat).any():
    failures.append(f"{sname}: uncovered positions remain NaN")
elif not torch.equal(mflat.view(torch.int16), full.view(-1).view(torch.int16)):
    failures.append(f"{sname}: bitwise mismatch")
else:
    n_pass += 1
dist.barrier()
dist.destroy_process_group()
if rank == 0 and (failures or n_checked == 0):
    sys.exit(1)

```

verl/workers/engine/megatron/transformer_impl.py

引擎侧配套：索引构建传入 slot_cache, get_per_tensor_param_shard 对跨 stage 占位行产出空 shard。

```

def _mcore_export_index(self):
    """Build (once) the per-parameter delta export index: geometry specs and
    form-B probes derived from the bridge's conversion tasks."""
    index = getattr(self, "_delta_export_index", None)
    if index is None:
        from .delta_export import build_export_index

        assert not self.vanilla_bridge, (
            "megatron delta_sharded is built on Megatron-Bridge param mappings; "
            "the deprecated vanilla mbridge flavor is not supported"
        )
        assert self.peft_cls is None, "megatron delta_sharded does not support LoRA"
        # slot cache 必须先于 index 建立，因为 build_export_index 内部会
        # 通过 _exchange_slot_tables 预 seed 每行的全局 slot 表
        self._delta_slot_cache: dict = {}
        index = build_export_index(self.bridge, self.module, self._delta_slot_cache)
        self._delta_export_index = index
        self._delta_export_by_name = {rec.megatron_name: rec for rec in index}
    return index

def get_per_tensor_param_shard(self, **kwargs):
    """Yield each rank's *local* mcore shard `(name, local_flat_bf16, ShardSpec)`."""

```

```

Params owned by another pipeline stage yield an empty shard
(zero-count lockstep rows; see the index builder)."""
load_megatron_model_to_gpu(self.module, load_grad=False)
index = self._mcore_export_index()

def _gen():
    # 跨 stage 占位行 :param 为 None, 产出空张量让 delta 引擎走锁步路径
    empty = torch.empty(0, dtype=torch.bfloat16, device=get_device_id())
    for rec in index:
        if rec.param is None:
            yield rec.megatron_name, empty, rec.spec
            continue
        local = rec.param.data
        if local.is_floating_point() and local.dtype != torch.bfloat16:
            local = local.to(torch.bfloat16)
        yield rec.megatron_name, local.reshape(-1), rec.spec

    return _gen(), None

```

评论区精华

PR 本身没有技术 review 评论，wuxibin89 直接 APPROVED；唯一的 issue 评论来自 gemini-code-assist bot，内容是 Gemini Code Assist 停机通知，与代码无关。真正的技术交锋记录在第二个 commit 的修复说明中：235B TP4×PP8×EP4 幂等扫描失败，违规者全部是专家权重。作者定位到先于 PP 就存在的归错根因——"the engine's batched gather merges entries BY SLOT POSITION within each directory row and rank 0 names the merged pieces from ITS OWN slot list"，而 mcore expert 路径此前只缓存各自 ep-local 专家名，违反位置合并契约；最终以 row-keyed union slot tables 修复，并新增逐行表一致性检查。PR body 的性能备注也承认：PP 形状在 7B 上较 TP2×PP1 基线增加约 1.5-4s 稳态 uw（目录行数翻倍 + 每参数 WORLD gather），预期在大规模下摊薄，并注明若 235B 结果不支持该预期，后续会做 owner-subgroup + rank0 hybrid gather group。

- ep_size>1 专家权重归错根因与 row-keyed slot table 修复 (correctness): 改为按目录行（而非参数名）键控的联合 slot 表，按 rank 顺序跨 merge group union；EP oracle 矩阵（qwen3_moe EP2 × PP1/PP2/PP2+EDP2）45/45 bitwise + 逐行表一致，EP1 回归全绿。
- PP 稳态同步的性能权衡与后续混合 gather 组 (performance): 235B TP4×PP8×EP4 实测稳态 uw 97.5-111.1s vs 同形状 NCCL 220.5-236.8s (2.1-2.4x)，收益显著；混合 gather 组明确留作 follow-up。

风险与影响

- 风险：
 1. 通信拓扑热点：PP>1 时所有参数并入 WORLD group，gather 参与方从 tp/etp×ep 子组扩到全部 rank，全局 rank 0 承担所有行的主聚合。7B 小模型上稳态 uw 增加 1.5-4s（数据来自 PR body），大模型下虽有 2.1-2.4x 收益但 payload 达 20-22GB，通信组规模对扩展性的影响仍需观察。

2. 强依赖 bridge 全局目录语义: build_export_index 的正确性建立在 get_conversion_tasks 的确定性顺序、跨 pp allgather、tied embedding 去重和 param_weight=None 占位行为之上; Megatron-Bridge 升级若改变这些语义, 可能导致目录不同 rank 分歧或占位误判。
3. 新集合点风险: _exchange_slot_tables 在 index 构建时引入一次 allgather_object, 若某 rank 的目录行数与全局不一致 (例如 bridge 版本混用), 会直接卡死或产生未对齐的 slot 表; 测试虽覆盖了 row-count 分歧检查, 但那是验证脚本而非运行时保护。
4. 测试覆盖的分布: differential oracle 覆盖 qwen2/qwen2_tied/qwen3_moe/nemotron_h 的 PP/EP 矩阵且全绿, 但 e2e 验证 (7B、235B) 依赖人工跑通, 未进入常规 CI; _exchange_slot_tables 在 PP=1 时也执行, 属于对所有现有用户路径的行为变更, 回归风险集中在 slot cache 的预 seed 时序。- 影响: 对用户: Megatron 后端 + PP/VPP 布局使用 delta_sharded checkpoint engine 的用户首次获得稳态增量权重同步, 此前只能退化为 seed 全量同步或全量 NCCL 同步; 大 MoE 模型 (如 235B 官方尺寸) 权重同步时间缩短约 55-60%。对系统: PP>1 下同步拓扑简化为单 WORLD 组, 消除了多 stage 间 relay 的复杂度, 但把聚合压力集中到全局 rank 0, 小规模 PP 场景有轻微稳态开销增加。对团队: 为后续 owner-subgroup + rank0 hybrid gather group 优化铺平了道路, 且 differential oracle 扩展为可复用的 PP/EP 回归工具链, 后续 bridge 升级可直接套用。- 风险标记: 核心路径变更, PP>1 全参 WORLD 组 gather 通信热点, 依赖 bridge 全局目录语义, 新集合点 allgather_object 卡死风险

关联脉络

- PR #7181 [megatron] feat: delta_sharded on Megatron-Bridge param mappings (TP+EP, hybrid-Mamba): 本 PR 的直接上游: 其在 Megatron 后端引入 delta_sharded 稳态导出 (TP+EP), 本 PR 解除其 PP=1 断言, 是同一功能线的纵深扩展, 且同时改到 delta_export.py 与 transformer_impl.py。
- PR #7144 [ckpt, fsdp] feat: sharded delta block placements + backend-owned HF export (engine core + FSDP): 定义了 ShardSpec 的 contributes/block placement 契约与 delta 引擎核心, 本 PR 的 zero-count 锁步行与 slot table 交换正是沿该契约扩展。
- PR #7085 [veomni] feat: EP-aware sharded delta export (fused expert stacks): commit 2 在修复 ep_size>1 专家归错时明确对照 VeOmni converter 的全局 hf_slots 契约, 说明两后端共享同样的位置合并语义。