PR #2175 完整报告

THUDM/slime

Fix R3 for allgather_cp

合并时间：2026-07-03 18:26

原文链接：http://prhub.com.cn/THUDM/slime/pull/2175

执行摘要

- 一句话：修复 allgather_cp 模式下 routing replay 专家对齐逻辑
- 推荐动作：建议精读 cp_utils.py 中新增的 prepare_routed_experts_for_routing_replay 函数，其展示了在 CP+TP 并行下如何对齐 rollout 与训练阶段的专家索引。该设计模式值得复用。同时关注后续是否追加测试覆盖。

功能与动机

在 allgather_cp 模式下，原有的专家令牌对齐逻辑未考虑 CP（Context Parallel）分组，导致路由回放（Routing Replay）时专家索引错位，进而影响 MoE 模型训练正确性。PR 标题 'Fix R3 for allgather_cp' 直接点明了修复目标。

实现拆解

1. 提取公用填充函数 _pad_routed_experts：在 cp_utils.py 中新增 _pad_routed_experts，用于将专家索引张量按指定大小 padding，填充值循环取模 num_experts。该函数替代了 actor 中原有的 pad_func 闭包。
2. 新增核心对齐函数 prepare_routed_experts_for_routing_replay：同样位于 cp_utils.py，接收 rollout 产出的专家索引序列和 token 序列，返回对齐后的专家张量。函数内部首先对每个序列 padding 1 个 token，然后根据 allgather_cp 标志选择不同路径：
   - 若 allgather_cp=True：将所有序列拼接后，按 CP_size * pad_size 对齐 padding，然后 chunk 分到当前 CP rank。
   - 若 allgather_cp=False：沿用原有逻辑，对每个序列调用 slice_with_cp 后拼接，再按 pad_size 对齐 padding。 最后若 sequence_parallel=True，按 TP rank 切分序列。
3. 简化 actor.py 中的 fill_routing_replay：移除原有内联的 pad_func 定义、断言循环、手动的 padding 和 slice_with_cp 调用，直接调用 prepare_routed_experts_for_routing_replay。同时更新导入语句，从 cp_utils 导入新函数而非 slice_with_cp。
4. 移除不再需要的 slice_with_cp 导入：actor.py 中原来导入 slice_with_cp，现在仅需 prepare_routed_experts_for_routing_replay，因此从导入列表中删除 slice_with_cp。
5. 配置参数传递：新函数通过关键字参数接收 num_experts、data_pad_size_multiplier、sequence_parallel 和新增的 allgather_cp，这些参数从 self.args 直接传入，无需在函数内部再次获取。

关键文件：

- slime/backends/megatron_utils/cp_utils.py（模块 后端；类别 source；类型 core-logic；符号 _pad_routed_experts, prepare_routed_experts_for_routing_replay）：新增了两个核心函数 _pad_routed_experts 和 prepare_routed_experts_for_routing_replay，实现了 allgather_cp 分支的专家对齐逻辑，是整个修复的核心。
- slime/backends/megatron_utils/actor.py（模块 后端；类别 source；类型 core-logic；符号 fill_routing_replay）：重写了 fill_routing_replay 方法，移除大量内联逻辑并委托给 prepare_routed_experts_for_routing_replay，简化了代码。

关键符号：_pad_routed_experts, prepare_routed_experts_for_routing_replay, fill_routing_replay

关键源码片段

slime/backends/megatron_utils/cp_utils.py

新增了两个核心函数 _pad_routed_experts 和 prepare_routed_experts_for_routing_replay，实现了 allgather_cp 分支的专家对齐逻辑，是整个修复的核心。

```
# slime/backends/megatron_utils/cp_utils.py

def _pad_routed_experts(experts: torch.Tensor, pad: int, num_experts: int) -> torch.Tensor:
    # 如果不需要 padding 则直接返回
    if pad == 0:
        return experts
    _, num_layers, topk = experts.shape
    # 生成填充值：使用 arange 并取模 num_experts，确保填充的专家索引在有效范围内
    pad_experts = (
        torch.arange(
            pad * num_layers * topk,
            device=experts.device,
            dtype=experts.dtype,
        ).reshape((pad, num_layers, topk))
        % num_experts
    )
    # 拼接原张量与填充张量
    return torch.cat([experts, pad_experts], dim=0)


def prepare_routed_experts_for_routing_replay(
    rollout_routed_experts: Sequence[torch.Tensor],
    tokens: Sequence[torch.Tensor],
    *,
    num_experts: int,
    data_pad_size_multiplier: int,
    sequence_parallel: bool,
    allgather_cp: bool,
) -> torch.Tensor:
    # 对齐 rollout 产出的路由专家元数据与训练 token 布局
    assert len(rollout_routed_experts) == len(tokens)
```

```
    for experts, token_ids in zip(rollout_routed_experts, tokens, strict=False):
        # 专家数应比 token 数少 1（最后一个 token 不参与路由预测）
        assert experts.shape[0] == token_ids.shape[0] - 1

    # 先对每个序列的最后一个位置 padding 一个 expert
    padded_experts = [_pad_routed_experts(experts, 1, num_experts) for experts in rollout_
    routed_experts]
    pad_size = mpu.get_tensor_model_parallel_world_size() * data_pad_size_multiplier

    if allgather_cp:
        # allgather_cp 模式：所有序列拼接后，按 CP_size * pad_size 整体对齐，再分片到各 CP rank
        routed_experts = torch.cat(padded_experts, dim=0)
        cp_size = mpu.get_context_parallel_world_size()
        cp_rank = mpu.get_context_parallel_rank()
        global_pad_size = cp_size * pad_size
        pad = (global_pad_size - routed_experts.size(0) % global_pad_size) % global_pad_size
        routed_experts = _pad_routed_experts(routed_experts, pad, num_experts)
        routed_experts = routed_experts.chunk(cp_size, dim=0)[cp_rank]
    else:
        # 非 allgather_cp 模式：每个序列先 slice_with_cp 再拼接，最后按 pad_size 对齐
        routed_experts = [
            slice_with_cp(experts, lambda x, pad: _pad_routed_experts(x, pad, num_experts))
            for experts in padded_experts
        ]
        routed_experts = torch.cat(routed_experts, dim=0)
        pad = (pad_size - routed_experts.size(0) % pad_size) % pad_size
        routed_experts = _pad_routed_experts(routed_experts, pad, num_experts)

    if sequence_parallel:
        # 若开启序列并行，按 TP rank 切分序列维度
        tp_rank = mpu.get_tensor_model_parallel_rank()
        tp_size = mpu.get_tensor_model_parallel_world_size()
        seqlen = routed_experts.size(0)
        assert seqlen % tp_size == 0
        start = seqlen // tp_size * tp_rank
        end = seqlen // tp_size * (tp_rank + 1)
        routed_experts = routed_experts[start:end]

    return routed_experts
```

slime/backends/megatron_utils/actor.py

重写了 fill_routing_replay 方法，移除大量内联逻辑并委托给 prepare_routed_experts_for_routing_replay，简化了代码。

```
# slime/backends/megatron_utils/actor.py ( 改动后 )

def fill_routing_replay(self, data_iterator, num_microbatches, rollout_data):
    if "rollout_routed_experts" not in rollout_data:
        raise ValueError(
```

```
            "rollout_routed_experts is required in rollout_data when use_rollout_routing_replay is
            set."
        )

    from megatron.core.transformer.transformer_block import get_num_layers_to_build
    from megatron.core.transformer.transformer_layer import get_transformer_layer_offset
    from slime.utils.routing_replay import RoutingReplay

    for iterator in data_iterator:
        iterator.reset()

    # 循环处理每个 micro-batch
    for _ in range(sum(num_microbatches)):
        batch = data_iterator[0].get_next(["rollout_routed_experts", "tokens"])
        # 委托给 cp_utils 中的对齐函数，传入所需配置
        rollout_routed_experts = prepare_routed_experts_for_routing_replay(
            batch["rollout_routed_experts"],
            batch["tokens"],
            num_experts=self.args.num_experts,
            data_pad_size_multiplier=self.args.data_pad_size_multiplier,
            sequence_parallel=self.args.sequence_parallel,
            allgather_cp=self.args.allgather_cp,
        )

        routing_replay_offset = 0
        for vp_stage, model in enumerate(self.model):
            config = model.module.config
            num_layers_to_build = get_num_layers_to_build(config, vp_stage=vp_stage)
            offset = get_transformer_layer_offset(config, vp_stage=vp_stage)
            for layer_id in range(offset, offset + num_layers_to_build):
                # 跳过稠密层（非 MoE 层）
                if isinstance(config.moe_layer_freq, int):
                    if layer_id % config.moe_layer_freq != 0:
                        continue
                elif isinstance(config.moe_layer_freq, list):
                    assert len(config.moe_layer_freq) > 0
                    # ... 后续逻辑不变
```

评论区精华

本 PR 无公开 review 讨论，合并人即作者，直接合并。

- 暂无高价值评论线程

风险与影响

- 风险：1）修改涉及 fill_routing_replay 核心训练循环，若 prepare_routed_experts_for_routing_replay 存在逻辑错误，可能导致所有 MoE 模型训练失败。2）移除了 actor 中原有的断言与检查，若新函数未完全等价替代，可能遗漏错误。3）

未新增测试，依赖现有测试覆盖，需关注 CI 结果。4）allgather_cp 分支新增了 chunk 操作，若 CP size 不能整除会导致断言失败。

- 影响：直接影响使用 allgather_cp=True 且启用 use_rollout_routing_replay 的 MoE 模型训练，此前可能训练异常，修复后恢复正确。对非 allgather_cp 模式无影响。代码可维护性提升，后续 routing replay 逻辑统一在 cp_utils.py 中修改。影响范围为 Megatron 后端训练流程。
- 风险标记：核心路径变更，缺少测试覆盖，并行一致性

关联脉络

- PR #2173 [docker] Update SGLang patch for PD R3 routed experts: 同样涉及 routed experts 的验证与配置修改，与本文的 routing replay 对齐逻辑属于同一功能线。
- PR #2169 Merging profiling info into router: 重构了 sglang 补丁与 routing 相关逻辑，本文进一步将 routing replay 逻辑从 actor 提取到 cp_utils，延续了重构趋势。